REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-02-

02588

and reviewing
r Information

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. RE
		01 DEC 98 - 30 NOV 01

4. TITLE AND SUBTITLE
OPTIMIZING SIMULATORS: AN INTELLIGENT ANALYSIS TOOL FOR COMPLEX OPERATIONAL PROBLEMS

5. FUNDING NUMBERS
F49620-99-1-0054

6. AUTHOR(S)
WARREN POWELL

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
PRINCETON UNIVERSITY
DEPARTMENT OF OPERATIONS RESEARCH AND FINANCIAL ENGINEERING
PRINCETON, NJ 08544

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
801 N. Randolph Street Room 732
Arlington, VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-99-1-0054

11. SUPPLEMENTARY NOTES

20020909 129

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The optimizing simulator represents a class of simulation tools in which the analyst can control the level of intelligence by adding information classes to the decision function. For example, the current MASS/AMOS simulator for airlift operations uses a simple rule-based function that acts purely on what is known at the time the decision is made, without using any forecasts of future activities. This is the first information class. The other three are: forecasts of exogenous events (classical forecasting), forecasts of the impact of a decision now on the future state of the system (for example, the impact of flying a C-17 into Saudi Arabia) and expert knowledge (although not reflect in the costs, an expert might tell you never to fly a C-17 into Saudi Arabia, or that it is best to use C5's when moving a certain type of cargo). Our approach to simulation bridges the traditional gab between simulation and optimization, and at the same time between operations research (which uses cost-based decision functions) and artificial intelligence (which uses rule-based decision functions). These techniques encompasses the current methods used in MASS (and its latest version AMOS), and at the same time can compete with commercial linear programming packages (which are used to solve models such as NRMO, which formulate the airlift problem as a linear program). We also allow the user to specify desired behaviors in the form of simple, low-dimensional patterns, which produces behaviors that may not be captured by a cost function. In this way, we provide a bridge between cost-based operations research models, and rule-based AT techniques.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	57
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Castle Laboratory

Department of Operations Research and Financial Engineering
Princeton University

Final Project Report:

Optimizing Simulators: An Intelligent Analysis Tool for Complex Operational Problems

Principal Investigator:

Warren B. Powell
Princeton University
Department of Operations Research and Financial Engineering
Princeton, NJ 08544
powell@princeton.edu

Prepared for:

Air Force Office of Scientific Research
Grant: F49620-99-1-0054

February, 2002

Abstract

The Air Force undertakes a variety of analysis efforts to ensure that it has the right number and type of resources to complete missions in a timely and efficient manner. A good example is the airlift simulator used by the Air Mobility Command (formerly MASS, with a new version called AMOS in production). Simulators such as these are very flexible, able to handle a high level of detail, but with very little intelligence. A competing technology is optimization models which offer a high level of intelligence, but limited flexibility. In this research, we propose a continuum of simulators, which we call *optimizing simulators*, which span the simplistic rules used in MASS and AMOS, to a highly intelligent tool that can compete with the most advanced optimization models and algorithms. We have identified four classes of information, and simulators can be designed which use just one class, or all four. The technique has been shown to compete with commercial linear programming packages on special problem classes, and to outperform these systems when we wish to explicitly model uncertainty. We encourage the use of cost-based logic (in common with optimization models) but provide for the inclusion of simple rules in the form of low dimensional patterns that provide direct control over the behavior of the simulator. The end result is a rich family of analysis tools that can be tailored to the needs of specific problems.

Table of contents

1. Executive summary

The *optimizing simulator* represents a class of simulation tools in which the analyst can control the level of intelligence by adding information classes to the decision function. For example, the current MASS/AMOS simulator for airlift operations uses a simple rule-based function that acts purely on what is known at the time the decision is made, without using any forecasts of future activities. This is the first information class. The other three are: forecasts of exogenous events (classical forecasting), forecasts of the impact of a decision now on the future state of the system (for example, the impact of flying a C-17 into Saudi Arabia) and expert knowledge (although not reflect in the costs, an expert might tell you never to fly a C-17 into Saudi Arabia, or that it is best to use C-5's when moving a certain type of cargo).

Our approach to simulation bridges the traditional gab between simulation and optimization, and at the same time between operations research (which uses cost-based decision functions) and artificial intelligence (which uses rule-based decision functions). These techniques encompasses the current methods used in MASS (and its latest version, AMOS), and at the same time can compete with commercial linear programming packages (which are used to solve models such as NRMO, which formulate the airlift problem as a linear program). We also allow the user to specify desired behaviors in the form of simple, low-dimensional patterns, which produces behaviors that may not be captured by a cost function. In this way, we provide a bridge between cost-based operations research models, and rule-based AI techniques.

The research has been organized along four key dimensions:

- The mathematical modeling of complex operational problems. – We propose a new mathematical modeling framework, called a *dynamic resource transformation problem*, that captures multilayered resources, and explicit modeling of the organization and flow of decisions and information.

- The design of efficient methods to provide the highest quality solutions to these problems. – We have proposed a new class of adaptive learning algorithms based on separable, piecewise linear functions. We have proven convergence under special conditions, and demonstrated experimentally that they produce high quality solutions with much faster convergence than earlier approaches.

- The implementation of these models and algorithms in software. – We have developed a Java-based modeling library for the problem class. The library simplifies some of the new modeling principles that we have developed, such as hierarchical aggregation, information decomposition, and the handling of multiattribute, multilayered resources.

- The testing of these ideas both on the airlift mobility problem, but also several real industrial problems. – In addition to applying these ideas to the airlift

mobility problem, we have also implemented these techniques at major railroads and trucking companies.

Our research has produced a number of specific findings:

- We have shown that the optimizing simulator approach provides high solution quality (both in the laboratory and in field implementation) offering both intelligence and tremendous modeling flexibility. Applications based on the optimizing simulator concept are now in production at Yellow Freight, the Burlington Northern Sante Fe Railroad, and the Norfolk Southern Railroad. We also have a simulator of airlift operations based on this concept.

- We have introduced the SPAR algorithm which uses sequences of separable, piecewise linear functional approximations. We have shown that SPAR provides provably optimal convergence in special cases, and optimal or near-optimal solutions with fast convergence for more general problems that arise in practice. The strategy allows an optimizing simulator to produce results that match commercial solvers on deterministic problems, and outperform them, sometimes by wide margins, for problems where uncertainty plays an important role (for example, if a C-17 might fail at an airbase which does not have proper repair facilities for this type of aircraft, it would be better to use a different type of aircraft for that route). The practical benefits of this new class of algorithmic strategy is the ability to provide optimizing behavior in a simulation-based framework (which provides outstanding modeling flexibility)

- We devised a method for accelerating the performance of the SPAR algorithm for problems where there are many different types of resources, using hierarchical aggregation (we estimate the value of another resource at a detailed level, and then produce estimates at multiple levels of aggregation). The result is a method that provides faster convergence in the early iterations, and better limiting performance than any algorithm working at a fixed level of aggregation.

- We now have algorithms which combine traditional engineering cost functions with low-dimensional patterns (for example, a pattern might be "avoid sending C-17's into Saudi Arabia). These are easy to specify and allow the user to produce desired behaviors without time-consuming engineering of the cost model. It also allows behaviors to be specified with different degrees of desirability. Based on the theory of proximal point algorithms, this logic brings together cost-based operations research models with rule-based artificial intelligence. In industrial applications (Yellow Freight System, Norfolk Southern Railroad) this logic has proven instrumental.

- We have extended the application of approximate dynamic programming methods using functional approximations to high dimensional batch service problems (for example, when to release an aircraft carrying different types of freight to a destination). This research showed that we could find near-optimal solutions using an iterative learning approach. The approach holds promise for solving

large-scale dynamic network design problems using an optimizing simulator strategy. This problem class has completely resisted other algorithmic strategies.

- We have shown that using higher levels of intelligence while simulating the airlift mobility problem can produce faster throughputs. These rules would be relatively easy to implement in the new AMOS system.

- We have demonstrated the new applications can be rapidly modeled using the new DRiP Java modeling library. This was tested recently by developing both a driver management system for Yellow Freight and a car distribution model for Norfolk Southern Railroad.

- Our general modeling framework provided the foundation for the design of a very general-purpose diagnostic library called PILOTVIEW that allows analysts to not only see the results of a model, but to step inside a model and analyze individual decisions. PILOTVIEW has been critical in the rapid development, debugging and calibration of large scale models.

2. Overview of research activities

The focus of this research was modeling and analysis tools for complex operational problems. This research had four broad themes:

- The mathematical modeling of complex operational problems.
- The design of efficient methods to provide the highest quality solutions to these problems.
- The implementation of these models and algorithms in software.
- The testing of these ideas both on the airlift mobility problem, but also several real industrial problems.

These four themes span the process of modeling from basic formulation to final implementation. The remainder of this section provides a more detailed summary of activities undertaken within each theme, with references to the key supporting research articles.

2.1 Mathematical modeling of complex operational problems

It is standard practice when formulating problems as optimization models to express the problem mathematically. This is facilitated by the simplifying assumptions which are traditionally made when formulating optimization models. By contrast, it is rarely the case that people will express simulation models mathematically. Instead, verbal descriptions and flowcharts are more common.

We undertook the development of a new mathematical modeling framework, developing as much as possible on existing styles and standards. The result is the *dynamic resource transformation problem* which is a mathematical vocabulary for very complex problems such as those faced by the air force, railroads and large trucking companies. It draws heavily from the concepts in linear programming and simulation, but offers several new modeling concepts:

- Problem dimensions – The DRTP framework outlines three major dimensions, each of which have a specific list of subdimensions. While not all problems will exhibit all dimensions, the result is a checklist of issues to be addressed in any modeling problem that is more comprehensive than would arise when working either within a classical math programming framework, or using standard simulation practices.
- Resource layering – Standard models will capture the flow of a "resource" to serve tasks. The airlift problem spans the movement of aircraft, pilots, freight, special equipment, maintenance capabilities and fuel. It is sometimes necessary to make decisions about an aircraft, with a particular pilot and load of freight, at a

particular fuel level. The modeling of resource layering extends classical modeling constructs such as multicommodity flows, and formalizes in an elegant way the handling of complex attribute vectors.

- Modeling the organization and flow of information – Simulation models implicitly model information. Most optimization models simply ignore it, and subtle notational styles actually interfere with the proper modeling of information. Our modeling notation is explicitly designed to model the information content of decisions in an elegant way that is completely accessible to modelers with an engineering background (rather than a strong background in stochastic processes). We model the evolution of information over time, and the organization of information among different agents.

- The use of hierarchical aggregation in information representation – When we do not know something, we often resort to aggregation to obtain additional insights. Aggregation is a powerful and widely recognized tool, but models are typically formulated at a single level of aggregation. We represent complex resources at three (or more) levels of aggregation, depending on what we are doing. The result is simulations that can handle a high level of detail without becoming computationally intractable.

This framework has produced two new problem classes. The first is called the heterogeneous resource allocation problem, which has been presented in:

Powell, W.B., J. Shapiro and H. P. Simao, "An Adaptive, Dynamic Programming Algorithm for the Heterogeneous Resource Allocation Problem," Transportation Science, Vol. 36, No. 2, pp. 231-249 (2002).

This problem class has proven useful in modeling the flows of relatively complex resources which might include people or equipment such as aircraft or locomotives.

The second is called the multilayered resource scheduling problem. A paper on this topic is in preparation.

The DRTP modeling framework is published in:

Powell, W.B., J. Shapiro and H.P. Simao, "A Representational Paradigm for Dynamic Resource Transformation Problems," Annals of Operations Research on Modeling (C. Coullard, R. Fourer, and J. H. Owen, eds), Vol. 104, pp. 231-279, 2001.

2.2 Algorithmic strategies

We have pursued a strategy that we call *optimizing simulators* which step through time making decisions, and then repeating this process, iteratively learning from prior

mistakes. This strategy is not new, but in the past has been restricted to relatively simple problems. We have devised strategies to apply this learning principle to very large scale problems.

There are several dimensions to our algorithmic strategy. The first begins with the basic strategy of the optimizing simulator, which may use four information classes. Two of the classes require that the simulator run iteratively. The first of these strategies arises when we want to learn the value of being in a particular state, and the second learns the number of times we take particular actions (which may or may not be deemed desirable). The process of learning the value of being in a state requires using a nonstandard dynamic programming recursion, summarized in section 1.2.1 below. Sections 1.2.2 and 1.2.3 outline activities for two major classes of resource allocation problems: those that are typically characterized by a small number of resource attributes, and discrete 0/1 problems which typically have large state spaces. Finally, section 1.2.4 describes methods for incorporating expert knowledge through low-dimensional patterns.

2.2.1 An adaptive learning strategy

Standard simulations step forward through time, applying rules to make decisions, and then applying the laws of system dynamics to describe the evolution of the state of the system. A more intelligent simulator uses information about the value of making a decision that puts the system into a particular state. The challenge is determining how to learn this value, and given this estimate, how to make decisions now in a computationally tractable way.

Textbook dynamic programming suggests estimating the value of being in a state using the recursion:

$$V_t^{n+1}(S_t) = \min_{x_t \in X_t(S_t)} c_t x_t + E\left\{V_{t+1}(S_{t+1}(x_t)) \mid S_t\right\}$$

It is well known that this approach suffers from the "curse of dimensionality" which arises when the state variable has multiple dimensions. Our problem actually suffers from three curses of dimensionality: the state space, the outcome space (which complicates our ability to find the expectation) and the action space (which complicates the problem of actually finding the best solution). Our approach is to use an *incomplete* state variable, which measures the state of the system immediately after a decision is made, but before new information is added. When we use this type of state variable, we obtain a recursion of the form:

$$V_t(S_t) = E\left\{\min_{x_t \in X_t(S_t)} c_t x_t + V_{t+1}(S_{t+1}(x_t)) \mid S_t\right\}$$

We then solve our three curses of dimensionality by a) dropping the expectation and solving the equation for a single sample realization, b) replacing the value function with an appropriately chosen functional approximation.

This strategy is used in several papers:

Godfrey, G. and W.B. Powell, "An Adaptive Dynamic Programming Algorithm for Single-Period Fleet Management Problems I: Single Period Travel Times," Transportation Science, Vol. 36, No. 1, pp. 21-39 (2002).

Papadaki, K. and W.B. Powell, "A Monotone Adaptive Dynamic Programming Algorithm for a Stochastic Batch Service Problem" European Journal of Operational Research, Vol. 142, No. 1, pp. 108-127 (2002).

Powell, W.B., H. Topaloglu and B. van Roy, "Approximate Dynamic Programming for Dynamic Resource Allocation: Merging Stochastic Programming and Dynamic Programming", in preparation.

2.2.2 Approximate dynamic programming for resource allocation problems

Approximate dynamic programming refers broadly to a class of techniques which approximate the future value of being in a future state. In contrast with classical techniques which step backward through time (and require enumerating the entire state space), these methods step forward through time, generating only a subset of states. However, they still struggle with problems with large state and action spaces (which includes virtually any resource allocation problem).

We solved the state space problem by using specially designed continuous

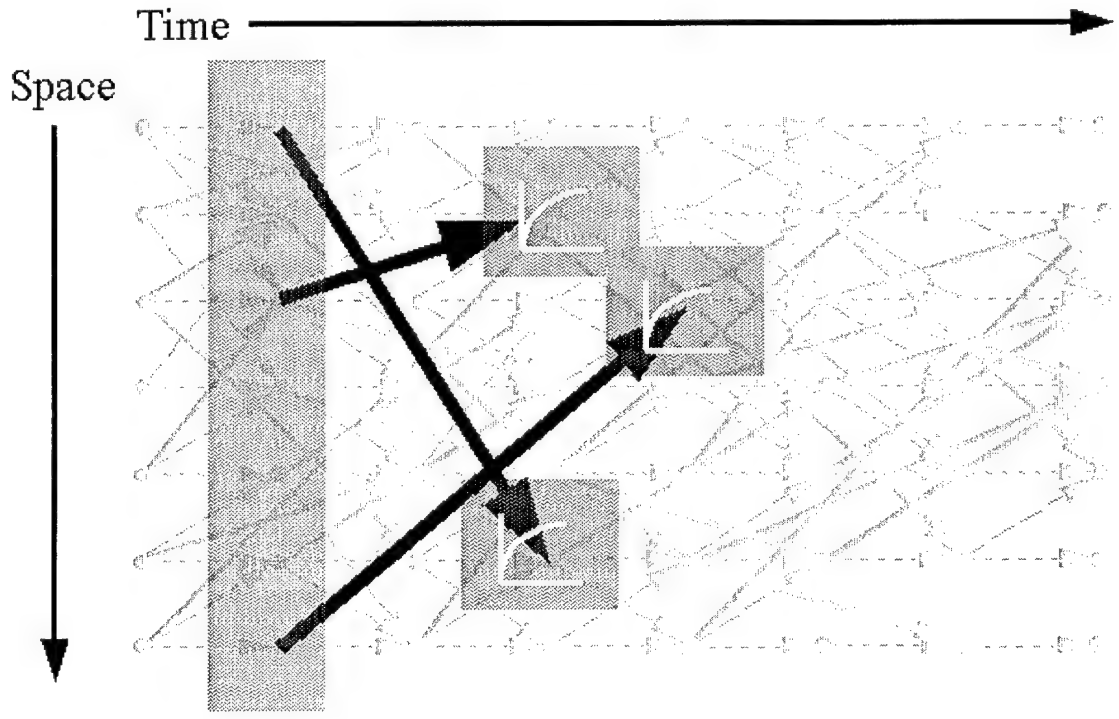

Figure 0 – Illustration of time-space network (in gray) and a subproblem at a point in time (shaded) using functional approximations of the future.

approximations. Linear approximations are the easiest to use, but a special class of separable, nonlinear approximations works the best, with faster convergence and better stability. The concept is illustrated in the figure 1, which illustrates a classical space time diagram (in gray) overlaid by shaded areas which illustrate solving the problem at time t, using nonlinear approximations of the future. When these approximations are separable, the resulting problem is quite easy to solve.

The nonlinear functions are created by sampling estimates of the value of one more unit of a particular type of a resource in the future. Repeating this process iteratively allows us to form sequences of piecewise linear, concave functions which express the value of additional units of resource as a function of the number of resources. The overall strategy requires solving sequences of fairly small linear programs using a commercial solver which also return the value of additional resources.

Because we sample from probability distributions as we step forward in time, these marginal values can be quite random, and we encounter the technical challenge of ensuring that our approximations are concave at *every* iteration. A strategy for accomplishing this was first proposed in:

Godfrey, G. and W.B. Powell, "An Adaptive, Distribution-Free Algorithm for the Newsvendor Problem with Censored Demands, with Application to Inventory and Distribution Problems," Management Science, Vol. 47, No. 8, pp. 1101-1112, (2001).

A variation of the algorithm, called a *leveling procedure,* was proposed with a proof of convergence in:

Topaloglu, H. and W.B. Powell, "An Algorithm for Approximating Piecewise Linear Concave Functions from Sample Gradients," submitted to Operations Research Letters.

A version of the algorithm called the SHAPE algorithm was posed and proven to be convergent for problems where the functions are continuously differentiable:

R. K.-L. Cheung and W.B. Powell, "SHAPE – A Stochastic, Hybrid Approximation Procedure for Two-Stage Stochastic Programs," Operations Research, Vol. 48, No. 1, pp. 73-79 (2000)

Experimental evidence demonstrated the effectiveness of the procedure in fleet management problems:

Godfrey, G. and W.B. Powell, "An Adaptive Dynamic Programming Algorithm for Single-Period Fleet Management Problems I: Single Period Travel Times," Transportation Science, Vol. 36, No. 1, pp. 21-39 (2002).

Godfrey, G. and W.B. Powell, "An Adaptive Dynamic Programming Algorithm for Single-Period Fleet Management Problems II: Multiperiod Travel Times," Transportation Science, Vol. 36, No. 1, pp. 40-54 (2002).

These experiments were applied to relatively simple flow problems known as single commodity flows (all flows are the same type of equipment). The work was extended to multicommodity flows in:

Topaloglu, H. and W.B. Powell, "Dynamic Programming Approximations for Stochastic, Time-Staged Integer Multicommodity Flow Problems," under revision for resubmission to Operations Research.

Multicommodity flow problems are somewhat simpler than heterogeneous resource allocation problems, but offer special complications when we are interested in integer solutions.

In many real problems, there is not a single decision maker, but instead several controllers (and possibly one at each geographical location). It is well known that we can decompose large problems into sequences of smaller problems, and coordinate these components through pricing mechanisms, which is equivalent to approximating other subproblems using linear approximations. In practice, these can be unstable. We adapted our nonlinear strategies to a multiagent setting, and found that even for multicommodity problems, each subproblem was a pure network. This work is summarized in:

Topaloglu, H. and W.B. Powell, "A Multi-Agent Decision Making Structure for Dynamic Resource Allocation with Nonlinear Functional Approximations," submitted to Operations Research.

Shapiro, J. and W.B. Powell, "A Metastrategy for Dynamic Resource Management Problems based on Informational Decomposition," submitted to Informs Journal on Computing.

One of the challenges of solving stochastic resource allocation problems using approximate methods is evaluating the quality of the solution. A popular technique for solving multistage stochastic resource allocation problems is Benders decomposition, which has proven convergence properties for two-stage problems but had not previously been proven for multistage problems. The following paper provides a variation of this algorithm for multistage problems:

Chen, Z.-L. and W.B. Powell, "A Convergent Cutting Plane and Partial Sampling Algorithm for Multistage Linear Programs with Recourse," Journal of Optimization Theory and Applications, Vol. 102, No. 3, pp. 497-524 (1999).

We are actively using this technique to evaluate the quality of our approximations for multistage problems.

2.2.3 Adaptive learning algorithms for discrete routing and scheduling

A separate line of research has been underway for discrete routing and scheduling problems, which offer special challenges. Nonlinear approximations are of little value when the flows are 0/1, but routing and scheduling problems tend to have very large state spaces for individual resources.

There is a vast array of algorithms for deterministic routing and scheduling problems, but the literature for dynamic problems is relatively young. A major challenge is that these problems have to be solved in real time. A paper that overcomes this hurdle, while still being able to handle the complex attributes of real drivers, is:

Powell, W.B., W. Snow and R. K.-M. Cheung, "Adaptive Labeling Algorithms for the Dynamic Assignment Problem," Transportation Science, Vol. 34, No. 1, pp. 67-85 (2000)

An often overlooked source of uncertainty is whether a human will actually implement a solution recommended by a model. A simulator showed that a very simple strategy for discounting dual variables would produce solutions that were better than algorithms that produced globally optimal solutions at each point in time, and better than greedy, myopic algorithms. This work is reported in:

Powell, W.B., M.T. Towns and A. Marar, "On the Value of Globally Optimal Solutions for Dynamic Routing and Scheduling Problems," Transportation Science, Vol. 34, No. 1, pp. 50-66 (2000).

Several papers were produced on a related problem in the area of parallel machine scheduling which extended the use of column generation strategies:

Chen, Z.-L. and W.B. Powell, "A Column-Generation Based Decomposition Algorithm for a Parallel Machine Just-In-Time Scheduling Problem," European Journal of Operations Research, Vol. 116, pp. 220-232 (1999).

Chen, Z.-L. and W.B. Powell, "Solving Parallel Machine Scheduling Problems by Column Generation," Informs Journal of Computing, Vol. 11, No. 1, pp. 78-94, Winter 1999.

Perhaps our most through investigation of a discrete routing and scheduling problem is our study of the dynamic assignment problem, which involves dynamically assigning resources to tasks over time. This problem is fundamental to all resource allocation problems, and represented our first attempt to study approximations for dynamic *two-layer* resource allocation problems. This work is reported in:

Spivey, M. and W.B. Powell, "The Dynamic Assignment Problem," submitted to Transportation Science, June, 2002.

2.2.4 Modeling expert knowledge

In real problems, it is almost always the case that a domain expert can look at the results of the model and criticize them. Most of the time it is criticizing activities that should not be done, but it may also be of the form "you should do this." These behaviors are easily coded into simulators as rules, but the strategy with optimization models is that we should cost out these activities. In practice, identifying all these costs is typically impractical, and the result is a model that "behaves badly" and which is hard to fix.

Overlooked in the criticism of optimization models is that many of these rules are complex and hard to adapt to new situations.

One strategy is to incorporate "rules" in the form of low dimensional patterns. A rule can be viewed as an action a that is used when a system is in state s. We can code a set of rules as a series of state action pairs of the form (s,a). For our problem classes, there can be an exponentially large number of states and actions.

Instead, we can focus on relatively simple rules that guide the optimization rather than tell us exactly what to do. The state of a system s includes, in principle, the state of every resource in the system. We avoid this complexity by restricting our attention to the attribute vector a of a single resource, and a decision d that will act on the resource. We may even represent this state/action pair at an aggregated level. We then let ρ_{ad} be a measure (for example, a percentage) of how often we want to use decision d on a resource with attribute a. Let R_a be the number of resources with attribute a, which means that $R_a\rho_{ad}$ is a measure of the number of resources with attribute a that we would expect to act on with decision d. If x_{ad} is the number of times our optimization model would like to act on a resource with attribute a with decision d, then $\sum_a\sum_d\left(x_{ad}-R_a\rho_{ad}\right)^2$ is a measure of the deviation between what we would like to do and what we are doing. Let $H(x,\rho)=\sum_a\sum_d\left(x_{ad}-R_a\rho_{ad}\right)^2$ be our "happiness function" which expresses the degree to which our optimization algorithm is matching desired patterns. We can then add this term to our engineering costs, multiplied by a scaling factor, to produce a modified objective function:

$$\min cx+\theta H(x,\rho)$$

This approach neatly combines simple rules with engineering costs. The patterns ρ can be specified by an expert, or from historical activities. We have found that both are useful.

This algorithmic strategy is known in the mathematical programming community as a proximal point algorithm. Using this approach to merge pattern matching with engineering costs appears to be new, and has proven to be one of the most powerful algorithmic and modeling strategies in our industrial applications. This strategy and several variants are summarized in:

Marar, A. and W.B. Powell, "Solving Resource Allocation Problems with Incomplete Information," under revision for resubmission to Management Science.

Marar, A. and W.B. Powell, "Capturing Incomplete Information in Resource Allocation Problems using Numerical Patterns", submitted to Operations Research.

2.3 Software implementation

Our work in software has been focused along two dimensions: the development of a general modeling library for dynamic resource allocation problems, and the development of a general purpose diagnostic tool for helping to identify problems with the model.

2.3.1 The DRiP Java modeling library

Developing simulators means writing code, and it is difficult for new programmers to appreciate all the dimensions of the problem. For example, a common mistake is to model something as a multicommodity flow problem initially, only to learn later that it should have been modeled as a heterogeneous resource allocation problem (which involves a major change in data structures). Modeling the availability of information is even more subtle. It is very easy for a programmer to implicitly force simplifying assumptions on internal data structures before realizing the implications.

It is possible to overcome some of these problems by imposing a modeling library on the software development effort. New programmers are forced to adopt certain modeling conventions that will prove useful as the project matures.

Efforts in this direction produced a java-based library that we refer to as the DRiP Java modeling library. The library is fully described in:

> *Joel Shapir,o "A Framework for Representing and Solving Dynamic Resource Transformation Problems,"Ph.D. dissertation Department of Operations Research and Financial Engineering, Princeton University, 1999.*

Special attention was given to how information is modeled:

> *Shapiro, J, W.B. Powell and D.E. Bernstein, "A Flexible Java Representation for Uncertainty in Online Operations Research Models," Journal of Computing, Vol. 13, No. 1, pp. 29-55, 2001.*

2.3.2 The PILOTVIEW diagnostic system

Once a model is up and running, we encounter the problem of diagnostics.

We have found that when we model complex systems, we often do not understand "why the model did that." Furthermore, we also typically find that when we theorize why the model behaves in a certain (usually undesirable way), that we are usually wrong.

Simulations can provide errant results because of one of four reasons: 1) the data is wrong, 2) the model is wrong (for example, the costs of decisions), 3) there is a flaw in the algorithm, or 4) there is a bug in the software. The problem is that identifying which of these is the culprit for a particular behavior is very difficult.

We have developed a general purpose diagnostic system called PILOTVIEW. This system is designed to work on general, multi-layer, multiattribute dynamic resource transformation problems. As a result, we are able to apply the system to problems involving aircraft, trucks or trains. We can model two-layer problems (aircraft and requirements; drivers and loads; locomotives and trains) or multilayer problems (at one company, we are modeling five layers: driver, tractor, trailer, product and customer).

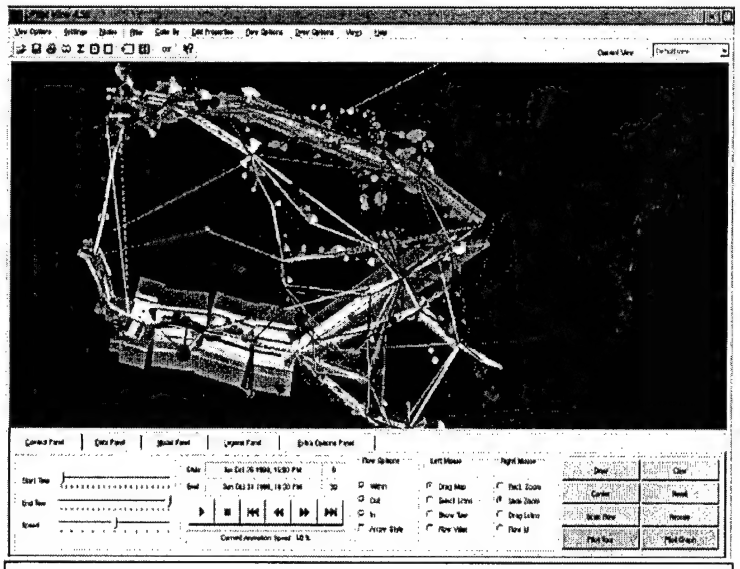

Figure 2 – Flows over an interval of time

PILOTVIEW operates on two types of datasets. The first is an activity file that gives the flows of resources (of different types, with different attributes) over time. There are three basic views for this data file: a static view (plots of flows over an interval of time, shown in figure 2), an animation of activities (showing objects moving over a map over time – Figure 3), and a graph of activities plotted over the entire horizon of the simulation.

Perhaps the most powerful tool within PILOTVIEW is the "Pilottour" module, which brings us right inside the optimization model itself. This is the tool that allows us to understand why the model made a particular decision. Pilottour works with a representation that a "resource" will have a vector of attributes "a", where there may be more than one resource with the same set of attributes. Pilottour displays "informational subproblems" which represent blocks of information that make up a single subproblem. Geographically, an informational subproblem can be the entire system (at one point in time), a single location, or a geographical area. Figure 4 illustrates a number of different

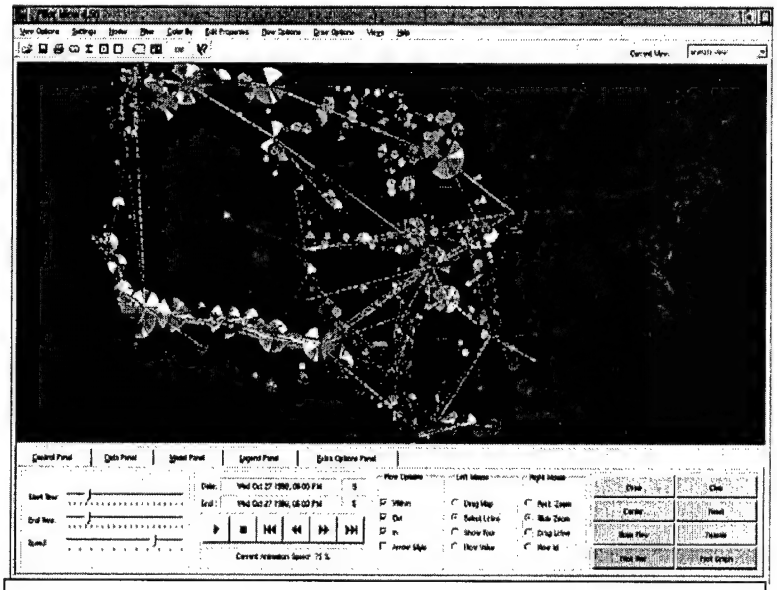

Figure 3 – Snapshot of animation, used to show actual movements.

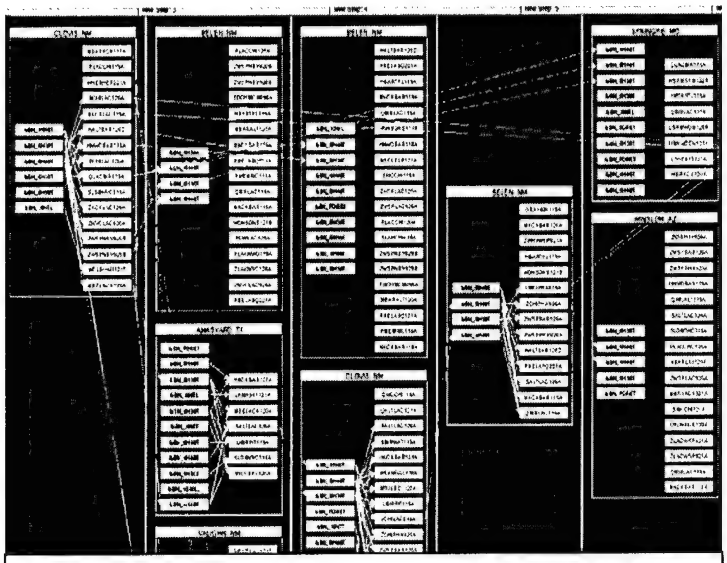

Figure 4 – The PILOTTOUR module, showing the inside of an optimization model, where the user can examine both the decisions that were made, and the ones that were not.

subproblems, each shown as a box with two columns of boxes (each column representing a resource layer). For the airlift problem, the first column represents types of aircraft, while the second column is requirements). Pilottour allows the analyst to click on any box or line (representing a possible decision) to obtain drill-down information. We have found that this tool allows people other than the original programmer of the model to analyze and diagnose problems.

2.4 Baby MASS

"Baby MASS" is a skeleton version of the airlift mobility simulator within MASS. This simulator models the assignment of aircraft to requirements, which are specified in a "TPFDD" (pronounced "Tipfid", TPFDD stands for "time phased, force deployment datasets" and contains the requirements that have to be moved. Each requirement represents an amount of freight (and possibly passengers), along with the time at which it has to be moved, the origin and destination, and the characteristics of the requirement (size and weight).

Baby MASS models two resource classes: aircraft and requirements. At this stage, aircraft are represented completely by their current location and the type of aircraft (which brings with it the capacity characteristics of the aircraft). The requirements are captured by their weight, origin and destination.

Once an aircraft is assigned to a requirement, it must move through a series of intermediate airbases. Unlike the production MASS simulator, we do not immediately move the aircraft (loaded with freight) through all the intermediate airbases to the destination. We simply move it to the first airbase. At that point, we capture the resource *layer* consisting of the aircraft loaded with the requirement. Although the logic is not in the code right now, we could model adaptive routing, where we change the path of the aircraft enroute as new information becomes available.

A major issue in the MASS simulator is airbase capacity. We also model airbase capacity as a soft constraint, in that we assess an increasingly high cost when an aircraft

passes through an airbase when the number of aircraft exceeds a target capacity. If we push an aircraft through an airbase that is over "capacity" (which itself is not a hard number), we pick up the cost of this decision, which can be used in the next iteration of the simulator. The idea is that if the airbase is heavily congested for a period of time during the simulation, the model, in the next iteration, should try to avoid this particular airbase.

Our decision on how to route the aircraft does not depend, then, on a simulation of the entire route of the aircraft at the moment that it first takes off. Instead, we simulate the path forward in time, and "discover" congestion as it occurs. The congestion is transmitted background through the simulation through the value function approximations, which indicate when a particular decision (such as assigning a particular type of aircraft to a requirement that involves using a congested airbase) is a poor one. This logic also allows the modeling of noise, such as a random aircraft failure on a runway that creates a high level of congestion (but not all of the time). Such a random event would not be known at the time that the aircraft is assigned, but it would be nice to see the model exhibit the behavior of avoiding airbases that are near capacity, simply because they are at greater risk of a high congestion cost as a result of a random failure. We can also capture the ability of the system to dynamically reroute aircraft if information becomes available after the aircraft has been loaded with freight to a particular destination.

We have performed comparisons between three types of simulations:

1. The original "MASS" logic of assigning the first available aircraft to the first available requirement.

2. Optimizing the assignment of aircraft to requirements at a particular point in time t, allowing us to choose the best aircraft for a particular assignment, but without considering any downstream effects.

3. Optimizing the assignment of aircraft to requirements, using value function approximations to approximate downstream behavior.

The first two options are both myopic simulators which are completed with a single pass. The third option requires the iterative training of adaptive dynamic programming. The third option is the only one that exhibits the behavior of moving an aircraft back to a home base (where requirements may originate) before a requirement has materialized (because any dispatcher would know that once the aircraft has emptied out, that it is of no value at its location in the theatre that the freight is being delivered to).

Our test dataset was a fairly small scenario that is not classified. Unfortunately, it is also not an especially rich dataset, with fairly simple requirements. We were able to show that the two optimizing runs (2 and 3) produced somewhat higher throughputs than the standard MASS run. The adaptive logic shows that it could produce overall faster throughput, without requiring the logic to "peek forward" to see if an airbase would be at

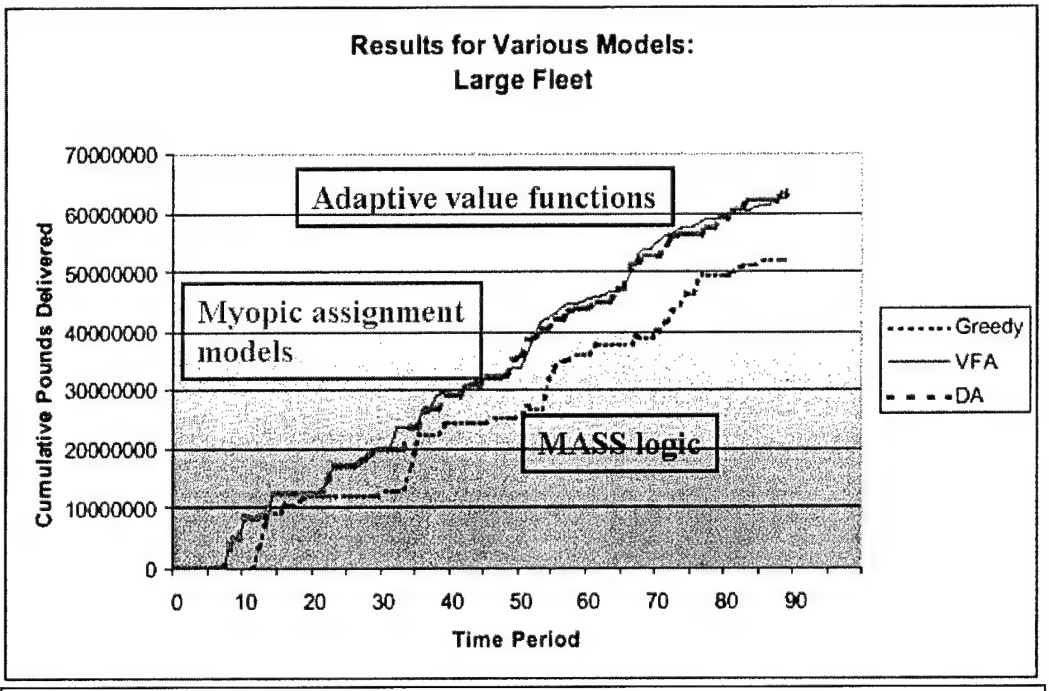

Figure 5 – Througput for airlift simulator using different levels of intelligence. Experiments were run on a small, unclassified dataset.

or over capacity.

The Baby MASS simulator has shown that the adaptive dynamic programming logic can work for the airlift problem. The test dataset provided a limited opportunity to show that the logic would produce faster throughputs. The intelligence behind the simulator offers the potential of producing more realistic simulations without requiring as many hard-coded rules to produce proper behaviors. The system can handle airbase congestion, without requiring either hard capacity constraints, or the need to simulate forward in time to estimate congestion. This logic is more amenable to modeling uncertainty in delays (due to airbase problems, or flight delays due to weather) since uncertainty is naturally captured in the value functions.

3. Research reports sponsored by AFOSR

The papers below summarize the research conducted in the broad area of resource management, with a focus on a) modeling complex problems and b) solving dynamic problems. Our research can be roughly divided along the following lines:

Section 3.1: Problem representation – These papers trace the evolution of our development of the dynamic resource transformation problem, which is a notational framework for a broad range of problems in resource management.

Section 3.2: Algorithms for dynamic routing and scheduling problems – These papers trace our work on the development of linear and nonlinear functional approximations for general resource management problems (single and multicommodity flow problems, and the heterogeneous resource allocation problem), and discrete routing and scheduling problems (where we have made contributions for both deterministic and stochastic problems).

Section 3.3: Modeling the organization and flow of information – One of the principles that have emerged from our research is the importance of modeling information: how it is organized (leading to multi-agent formulations), how it flows (leading to stochastic problems), and how it is represented (which brings us to the problem of optimizing problems with incomplete information).

Section 3.4: Software architecture – Complex problems need to be modeled in software. We have developed the DRiP Java modeling library, which is a Java-based library built around the principle of the optimizing simulator. The library has also been designed to capture some of the important characteristics of the DRTP modeling paradigm.

Section 3.5: Implementation research – Although the interest in the air mobility command is on simulation and analysis, we have found significant interest in the implementation of dynamic models in production at companies that have sponsored Castle Laboratory. Here we summarize both laboratory and field research that is focused on issues that arose in field implementations.

Section 3.6 Student theses – The grant has provided direct support of the research of several Ph.D.'s and masters candidates, as well as indirect support for several undergraduate senior theses.

3.1 Problem representation

Powell, W.B., J. Shapiro and H.P. Simao, "A Representational Paradigm for Dynamic Resource Transformation Problems," Annals of Operations Research on Modeling (C. Coullard, R. Fourer, and J. H. Owen, eds), Vol. 104, pp. 231-279, 2001.

"Toward a Unified Framework for Real-Time Logistics Control," Military Operations Research, Vol. I, No. 4, Winter, 1996, pp. 69-79.

Powell, W.B. "On Languages for Dynamic Resource Scheduling Problems," in Fleet Management and Logistics, (T. G. Crainic and G. Laporte, eds.), Kluwer Academic Publishers, Boston, 1998.

3.2 Algorithms for dynamic routing and scheduling problems

9.2.1 Linear value function approximations

Powell, W.B., J. Shapiro and H. P. Simao, "An Adaptive, Dynamic Programming Algorithm for the Heterogeneous Resource Allocation Problem," Transportation Science, Vol. 36, No. 2, pp. 231-249 (2002).

Powell, W.B. and T. Carvalho, "Dynamic Control of Logistics Queueing Networks for Large Scale Fleet Management," Transportation Science, Vol. 32, No. 2, pp. 90-109, 1998.

Carvalho, T. and W.B. Powell, "A Multiplier Adjustment Method for Dynamic Resource Allocation Problems," Transportation Science, Vol. 34, No. 2, pp. 150-164 (2000).

R. K.-L. Cheung and W.B. Powell, "SHAPE – A Stochastic, Hybrid Approximation Procedure for Two-Stage Stochastic Programs," Operations Research, Vol. 48, No. 1, pp. 73-79 (2000)

Powell, W.B., W. Snow and R. K.-M. Cheung, "Adaptive Labeling Algorithms for the Dynamic Assignment Problem," Transportation Science, Vol. 34, No. 1, pp. 67-85 (2000)

9.2.2 Nonlinear functional approximations

Powell, W.B., A. Ruszczynski, and H. Topaloglu, "Learning Algorithms for Separable Approximations for Stochastic Approximation Problems," under review at Mathematics of Operations Research.

Godfrey, G. and W.B. Powell, "An Adaptive, Distribution-Free Algorithm for the Newsvendor Problem with Censored Demands, with Application to Inventory and Distribution Problems," Management Science, Vol. 47, No. 8, pp. 1101-1112, (2001).

Chen, Z.-L. and W.B. Powell, "A Convergent Cutting Plane and Partial Sampling Algorithm for Multistage Linear Programs with Recourse," Journal of Optimization Theory and Applications, Vol. 102, No. 3, pp. 497-524 (1999).

Topaloglu, H. and W.B. Powell, "Dynamic Programming Approximations for Stochastic, Time-Staged Integer Multicommodity Flow Problems," under revision for resubmission to Operations Research.

Godfrey, G. and W.B. Powell, "An Adaptive Dynamic Programming Algorithm for Single-Period Fleet Management Problems I: Single Period Travel Times," Transportation Science, Vol. 36, No. 1, pp. 21-39 (2002).

Godfrey, G. and W.B. Powell, "An Adaptive Dynamic Programming Algorithm for Single-Period Fleet Management Problems II: Multiperiod Travel Times," Transportation Science, Vol. 36, No. 1, pp. 40-54 (2002).

9.2.3 Stochastic routing and scheduling

Spivey, M. and W.B. Powell, "The Dynamic Assignment Problem," under review at Transportation Science.

Powell, W.B., W. Snow and R. K.-M. Cheung, "Adaptive Labeling Algorithms for the Dynamic Assignment Problem," Transportation Science, Vol. 34, No. 1, pp. 67-85 (2000)

9.2.3 Deterministic routing and scheduling

Chen, Z.L. and W.B. Powell, "Exact Algorithms for Scheduling Multiple Families of Jobs on Parallel Machines," Naval Research Logistics. (to appear).

Chen, Z.-L. and W.B. Powell, "A Column-Generation Based Decomposition Algorithm for a Parallel Machine Just-In-Time Scheduling Problem," European Journal of Operations Research, Vol. 116, pp. 220-232 (1999).

Chen, Z.-L. and W.B. Powell, "Solving Parallel Machine Scheduling Problems by Column Generation," Informs Journal of Computing, Vol. 11, No. 1, pp. 78-94, Winter 1999.

9.2.4 Batch service processes

Papadaki, K. and W.B. Powell, "A Monotone Adaptive Dynamic Programming Algorithm for a Stochastic Batch Service Problem" European Journal of Operational Research, Vol. 142, No. 1, pp. 108-127 (2002).

Papadaki, K. and W.B. Powell, "An Adaptive Dynamic Programming Algorithm for a Stochastic Multiproduct Batch Dispatch Problem" under revision for resubmission to Naval Research Logistics.

Papadaki, K. and W.B. Powell, "A Discrete On-Line Monotone Estimation Algorithm" submitted to Annals of Mathematical Statistics.

3.3 Modeling the organization and flow of information

Topaloglu, H. and W.B. Powell, "A Multi-Agent Decision Making Structure for Dynamic Resource Allocation with Nonlinear Functional Approximations" under review at Operations Research.

Marar, A. and W.B. Powell, "Solving Resource Allocation Problems with Incomplete Information," under revision for resubmission to Management Science.

Powell, W.B., "Managing Information through the Control of Information," submitted to OR Chronicles, ~November, 2000. Under revision for resubmission to Operations Research.

Marar, A. and W.B. Powell, "Reducing the Optimality Gap for Dynamic Resource Allocation Problems using Information Representation" in preparation for submission.

Marar, A. and W.B. Powell, "An Algorithm for Minimizing the Pearson Goodness-of-Fit Measure as an Integer Dynamic Resource Allocation Problems" or "An Algorithm for Incorporating Incomplete Information in Dynamic Resource Allocation Problems using the Pearson Goodness-of-Fit Measure." In preparation for submission.

Marar, A. and W.B. Powell, "Representing Information from Regression Trees in Resource Allocation Problems," in preparation for submission.

Shapiro, J. and W.B. Powell, "A Metastrategy for Dynamic Resource Management Problems based on Informational Decomposition," under revision for resubmission to Informs Journal on Computing.

3.4 Software architecture

Shapiro, J, W.B. Powell and D.E. Bernstein, "A Flexible Java Representation for Uncertainty in Online Operations Research Models," Journal of Computing, Vol. 13, No. 1, pp. 29-55, 2001.

"An Object Architecture for Dynamic Resource Transformation Problems, (J. Shapiro and W.B. Powell). Working paper.

The DRiP Java modeling library: A Tutorial. Found on the web at:

http://www.castlelab.princeton.edu/DRTProot/DRTP/tutorial.html

3.5 Implementation research

Powell, W.B., A. Marar, J. Gelfand, and S. Bowers, "Implementing Operational Planning Models: A Case Application from the Motor Carrier Industry," Operations Research, Vol. 50, No. 4, pp. (2002).

Powell, W.B., M.T. Towns and A. Marar, "On the Value of Globally Optimal Solutions for Dynamic Routing and Scheduling Problems," Transportation Science, Vol. 34, No. 1, pp. 50-66 (2000).

3.6 Student theses

3.6.1 Doctoral dissertations

Katerina Papadaki, "Adaptive Dynamic Programming for Aging and Replenishment Processes," 2002.

Arun Marar, "Information Representation in Large-Scale Resource Allocation Problems: Theory, Algorithms and Applications," 2002.

Huseyin Topaloglu, "Dynamic Programming Approximations for Dynamic Resource Allocation Problems," 2001.

Mike Spivey, "The Dynamic Assignment Problem," 2001.

Joel Shapiro, "A Framework for Representing and Solving Dynamic Resource Transformation Problems," 1999.

3.6.2 Masters theses dissertations

Jayanth Marasanapalle, "Function Approximations for Integer, Stochastic Resource Allocation Problems." 2000.

3.6.3 Undergraduate senior theses

Edward Colburn, *The Optimization of Pricing Decisions Over a Dynamic Shipping Network Using Stochastic Gradient Algorithms*, 2002.

Vazquez-Gil, Xabier, *Learning to Fly: An Adaptive Dynamic Programming Approach for the Air Mobility Command Problem*, 2001.

Kevin White, A Report from the Flight Deck: An Empirical Analysis of the Fractional Jet Ownership Industry, 2000.

4. Personnel supported

Faculty:

Professor Warren B. Powell

Professor Andrzej Ruszczynski (visiting professor from RUTCOR).

Professional staff:

Dr. Hugo Simao

Dr. Belgacem Bouzaiene-Ayari

Doctoral students:

Tony Wu (3^{th} year)

Abraham George (2^{nd} year)

Katarina Papadaki (graduated 2002)

Michael Spivey (graduated 2002)

Arun Marar (graduated 2002)

Huseyin Topaloglu (graduated 2001)

Joel Shapiro (graduated 1999)

Masters students:

Jayanth Marasanapalle

5. Interactions/transitions

5.1. Participation/presentations at meetings, conferences, etc.

Invited talks:

"Real Time Optimization for Real-World Operations," Informs Practice Meeting, Montreal, May, 2002.

"Adaptive Dynamic Programming for Large-Scale Resource Allocation: Solving the three curses of dimensionality," NSF Workshop on Learning and Approximate Dynamic Programming and NSF Workshop on the Electric Power Industry, Mexico, April, 2002.

"The Optimizing Simulator: Raising the 'IQ' of Airlift Simulations," Informs Chapter presentation, Air Mobility Command, Scott Air Force Base, March, 2002.

"The Optimizing Simulator: Understanding Information in the Modeling of Airlift Operations," Minnowbrook Conference Center, November, 2001.

"An Information-Theoretic Approach to Solving the Locomotive Power Management Problem," University of Linkoping, Linkoping, Sweden, March, 2001.

"Modeling Information in Dynamic Resource Management," NJ Chapter of Informs Meeting, RUTCOR, February, 2001.

"Adaptive Dynamic Programming for Dynamic Resource Management," University of Chicago, Graduate School of Business, November, 2000.

"Information Theory in Resource Management," Seminar given at the Universite de Montreal, February, 2000.

"Tutorial on Dynamic Resource Management," Mathematisches Forschungsinstitut, Oberwolfach, Conference on Traffic and Transport Optimization, Germany, November, 1999.

"Optimization Models for the Motor Carrier Industry: An Emerging Information Technology," Truckload Motor Carriers Conference, Birmingham, September, 1999.

"Dynamic Programming Approximations in Multi-Stage Linear Programming," New World Vistas Conference, Sponsored by Air Force Office of Scientific Research, New York, May, 1999.

Plenary speech: "An Information Theoretic Approach to Dynamic Resource Management," Optimization Days, Montreal, Quebec, May, 1999.

"Dynamic Programming Approximations for Multicommodity Network Flows: Deterministic and Stochastic Problems," DIMACS Workshop on Logistics, Rutgers, February, 1999.

"Tutorial: Dynamic Optimization Models for Complex Operations," Seminar, University of Montreal, Montreal, Quebec, February, 1999.

Conference presentations:

"An Information Theoretic Model of Locomotive Operations," IFORS 2002, Scotland (with Belgacem Bouzaiene-Ayari).

"A Multilayered Resource Scheduling Problem," IFORS 2002, Scotland (with Hugo Simao and Raymond Cheung).

"An Information Theoretic Approach to Modeling Car Distribution," IFORS 2002, Scotland (with Huseyin Topaloglu and S. Melkote).

"Solving a Large Scale Driver Management Problem Using Informational Decomposition," IFORS 2002, Scotland (with Hugo P. Simao).

"Tutorial: Emerging Developments in Adaptive Dynamic Programming for Stochastic Resource Management," Informs National Meeting, Miami, November, 2001.

"A Multi-Agent Approach for Stochastic Multicommodity Flow Problems with Applications in Fleet Management," Informs National Meeting, Miami, November, 2001.

"A Dynamic Optimization Model for Locomotive Management based on Information Modeling," Informs National Meeting, Miami, November, 2001.

"Optimizing Complex Operational Problems under Incomplete Information, with an Application to Locomotive Power Management," TRISTAN IV, Azores, June, 2001 (with A. Marar and B. Bouzaiene-Ayari).

"The Multi-layered Resource Scheduling Problem," TRISTAN IV, Azores, June, 2001 (with H.P. Simao).

"Tutorial: Adaptive Dynamic Programming for Dynamic Resource Management," TRISTAN IV, Azores, June, 2001 (with H. Topaloglu).

"Dynamic Programming Approximations for Time-Staged Stochastic Integer Multicommodity Flow Problems," Informs National Meeting, San Antonio, November, 2001 (with H. Topaloglu).

"The Dynamic Assignment Problem," Informs National Meeting, San Antonio, November, 2001 (with M. Spivey).

"Adaptive Dynamic Programming for Real-Time Locomotive Management," Informs National Meeting, San Antonio, November, 2001 (with Belgacem Bouzaiene-Ayari).

"Adaptive Dynamic Programming Methods for Aging and Replenishment Problems with Applications to Inventory Management and Vehicle Dispatching," Informs National Meeting, San Antonio, November, 2001 (with K. Papadaki).

"Tutorial: Dynamic Resource Management – Problems, Models and Algorithms," Informs National Meeting, San Antonio, November, 2001.

"Dynamic Programming Approximations for the Dynamic Assignment Problem," Informs National Meeting, Salt Lake City, May, 2000 (with M. Spivey).

"Adaptive Dynamic Programming for Locomotive Scheduling," ," Informs National Meeting, Salt Lake City, May, 2000 (with Belgacem Bouzaiene-Ayari).

"Structural Dynamic Programming for Dynamic, Multicommodity Flow Problems in Transportation," Informs National Meeting, Salt Lake City, May, 2000 (with H. Topaloglu).

5.2. Consultative and advisory functions

I have given presentations each year on this research to the Air Mobility Command. I also met with Mike Strickland who is a consultant working on the new AMOS simulator, and we are trying to mimic the AMOS dispatch rules with one of our decision functions. This will allow us to make comparisons directly to the existing AMOS logic.

5.3. Transitions

Our transitions have occurred along three lines:

- Meetings with analysts at the Air Mobility Command and their subcontractors for rewriting the old MASS simulator (new name: AMOS).
- Direct implementation of ideas through projects with the corporate partners of CASTLE Lab.

- Licensing of software through local consulting firms for use in systems for their clients. CASTLE Lab has relationships with Transport Dynamics, Inc. (TDI), and Princeton Consultants, Inc. (PCI).

Specific transitions to the industrial partners of CASTLE Lab include:

1 Transition: Operational, tactical and strategic planning of locomotives. This system uses the optimizing simulator concept, and in particular makes heavy use of techniques for modeling incomplete information through low dimensional patterns. The system was recently approved for production at Norfolk Southern Railroad, making it the first successful production optimization model developed for operational use in North America.

Recipients: Norfolk Southern Railroad, which uses the system both for strategic planning of the fleet size, and short-term tactical forecasting of surpluses and deficits.

Burlington Northern Sante Fe Railroad, where the primary focus is on routing locomotives to maintenance facilities, where they have to arrive at a particular time.

2. Transition: We have been working for years to developing an operational routing and scheduling system for the multilayered resource scheduling problem. This has been applied to the routing and scheduling of drivers, tractors, trailers, product and customer tanks. This system is now in production and is available for schedulers to use on demand.

Recipient: Air Products and Chemicals

3. Transition: Forecasting of time series activities with multiple calendar effects using hierarchical aggregation.

Recipient: Yellow Freight System, Norfolk Southern Railroad

Transition: A driver scheduling system for large-scale less-than-truckload motor carriers using informational decomposition. We combined the adaptive learning techniques for coordinating multiple agents along with low dimensional patterns for capturing incomplete information. The system is now being used to perform short term tactical forecasts of movements of drivers.

Recipient: Yellow Freight System

4. Transition: A real-time routing and scheduling algorithm for short-haul truckload trucking using an adaptive labeling algorithm. This is the first real-time system which can produce tours (where a truck covers multiple tasks) running for fleets with several hundred trucks.

5. Transition:

   Recipient: Triple Crown Services.

Research results licensed to Transport Dynamics are now incorporated into optimization models TDI sells to truckload carriers (forecasting and real-time routing and scheduling) and less-than-truckload motor carriers (which requires forecasting as well as load consolidation). These systems are now helping to run FedEx Ground, FedEx Custom Critical, Watkins Motor Lines, and de Boer Truckline. These are now successful commercial product lines.

Princeton Consultants undertakes custom projects, and is currently involved in a project to implement an optimizing-simulator based solution to Donnelley, a $5 billion printing company.

Technical appendix

A.1 Overview

Computer models that simulate physical activities have long proven to be a powerful analysis tool for designing complex operations. For example, the MASS simulation system has been an effective tool for the analysis group at the Air Mobility Command to estimate throughput and response times, and to analyze the effects of different airbases, aircraft characteristics and aircraft routing policies on system performance. This analysis has traditionally been performed with a classical simulation model, developed over the years by Tom Kowalsky and recently updated by an outside contractor.

The power of simulation is its flexibility. Its weakness is that a) decisions are typically made using rule-based functions that often have to be redesigned to produce the right behaviors for a given dataset, and b) the simulator does not readily provide a measure of which resources are constraining the system. As a result, a fair amount of trial-and-error is needed to determine the changes in inputs that produce the most effective improvements to airlift responsiveness for a particular scenario. Classical simulators in particular struggle with problems in logistics where the dimensionality of a decision can be high. For example, in a manufacturing setting, it is often the case that "decisions" are very simplistic (for example, well-defined rules may determine which job is handled next by a particular machine). By contrast, logistics problems are often characterized by vectors of decisions (for example, which of 10 aircraft should handle which of 5 different requirements, producing a vector of decisions with 50 elements).

A competing technology is math programming (and specifically, linear programming) which is particularly adept at handling vectors of decisions. Furthermore, after finding the optimal solution, an LP solver will also produce dual variables for resource constraints, which provide an indication of which resources (aircraft, airbases, fuel, airspace capacity) would provide the biggest impact if they were increased. LP solvers also provide a sense of "optimality" by providing a formal method of determining the best set of decisions. The intelligence behind the optimization logic allows the system to adapt more quickly to different datasets. Optimization logic, however, requires formulating an "objective function" which provides the total cost for a particular set of decisions. The objective function provides a formal mechanism for determining when one decision is better than another, but the quality of the decisions depend, on course, on the degree to which the objective function accurately reflects desired behaviors. Often, the "costs" in an objective function are artificial bonuses and penalties designed to produce a specific behavior.

Just as important, optimization models require that the *dynamics* of the system be described using systems of linear equations. Engineers modeling complex systems often find it difficult to capture the rules that govern the evolution of real systems using systems of equations. It is typically far easier to capture the difficult physics of real operations using the type of rule-based logic that simulation models provide.

Needless to say, the pros and cons of each approach has spawned a healthy debate.

This document provides a summary of research that overcomes this debate through a concept we refer to as an "optimizing simulator." We are going to describe a technology that offers the flexibility of a simulator, and yet which can compete against linear programming solvers. As with optimization algorithms, we use a cost function to determine which simulation is best, but in contrast with the classical use of simulation, we provide an easy way of incorporating rule-based behavior (a merger of "AI" and "OR"). We do not have any difficulty handling the large dimensionality of decision vectors that typically characterize problems in logistics. Finally, we use the same rule-based logic for representing the evolution of system dynamics as is found in any simulation package.

A.2 The optimizing-simulator concept

The optimizing-simulator is based on traditional concepts of simulation. It is useful to first summarize the elements of a simulator (but in our notation), which we do using the airlift flow problem as an example. We then introduce how we would solve the airlift flow problem as an optimizing simulator.

A.2.1 The basic simulator

A traditional simulator steps through time, making decisions and modeling the evolution of the system as new events arise. The simplest simulators do not actually have to "make" decisions, but rather simply model the physics of the system. These systems evolve as a result of two inputs. The first we call "knowledge" which is information from an exogenous source. We let:

κ_t = The "knowledge" that arrives in time t.

Let:

K_t = The "knowledge base" at time t.

Given what we know (K_t) at time t, we next make decisions. Let:

x_t = The vector of decisions at time t.

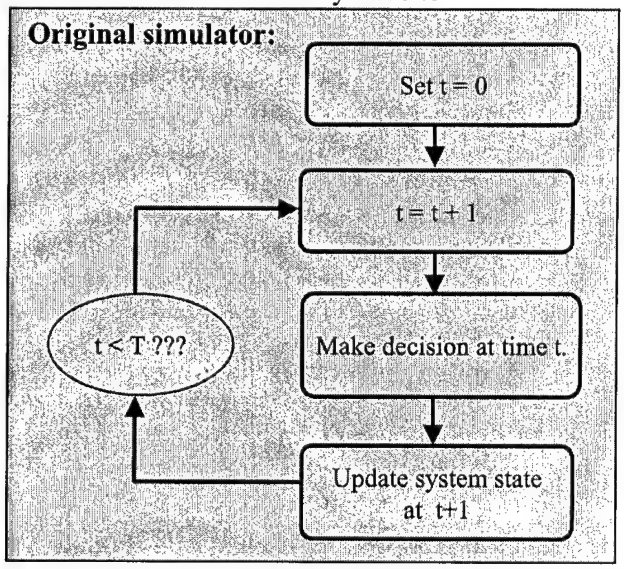

Figure A.1 – Illustration of classical simulation logic stepping through time.

It is useful to think of κ_t as (a random variable) representing *exogenous* information, while x_t is a decision, which represents *endogenously controllable* information. Our system evolves through a sequence of exogenous and endogenous information, which we may represent using:

$$(K_0, x_0, \kappa_1, x_1, \kappa_2, x_2, ...)$$

We are not going to worry about where the exogenous sequence $(\kappa_t)_{t\geq 0}$ comes from, but we are very interested in how we make decisions. We can represent the process of making decisions using a function that we denote by X^π, which we can think of as a mapping from what we know at time t to a vector of decisions. To capture what we know, assume that we have a *knowledge updating* function that handles the update of what we kow:

$$K_{t+1} \leftarrow U^K(K_t, x_t, \kappa_t)$$

Our decision function, then, is a mapping from what we know at time t to a set of decisions:

$$X^\pi(K_t) \rightarrow x_t$$

The evolution of the system can be completely specified through the knowledge update function $U^K()$ and the decision function $X^\pi()$. We assume that the knowledge updating function is specified as part of the problem. By contrast, there are many different ways of making decisions. For this reason, we define a set of *policies* Π, and let $(X^\pi), \pi \in \Pi$ to be a family of decision functions, and our challenge is finding the best function (or policy).

In classical simulators, both the knowledge update function and the decision function are typically rule-based. This is especially difficult when decisions come in vectors. Consider, for example, the situation faced by the airlift flow module of MASS, which is depicted (with a little imagination) in figure A.1, which shows a set of aircraft we can choose from on the left, and a set of requirements (such as freight) that has to be moved on the right. Each aircraft, and each requirement, has a set of attributes. An aircraft will have attributes such as location, capacity, ability to handle oversized freight or passengers, speed, range, and landing requirements (such as length of runway). Requirements have attributes such as origin and destination, weight and volume, oversize characteristics, passengers, and the time at which it is available and when it has to move. If we face the problem of choosing how to assign four aircraft to three requirements (keeping in mind that it is common to need more than one aircraft to move a requirement), we have $3^4 = 81$ possible decisions we can make (a number that grows quickly with the number of aircraft and requirements that we want to consider at one time). Not only do we need a way of determining when one decision is better than

another (an objective function) we also need a very fast way of sorting through these different combinations (enumerating them is not an option).

The original MASS simulator resolves this issue very simply. The aircraft and requirements are sorted in terms of order of availability. The system then takes the first aircraft that is available, and then chooses the first available requirement, and attempts to assign one to the other. The difficulty is determining whether this assignment is possible. The aircraft might be in, say, New Jersey, while the requirement may have to be moved from the Philippines to Southeast Asia. Getting the aircraft from New Jersey to the Philippines will require stopping and refueling at several intermediate airbases. These airbases may be at capacity. It is necessary to run a short simulation to determine when the aircraft will reach each airbase, and whether it has the capacity to handle the aircraft. "Capacity" is itself a fairly complex issue, since it may reflect the ability to land and store the aircraft, as well as refueling and maintenance capacity. If it is decided that the assignment is possible, then this decision is made. There is no attempt to consider whether any other decisions are possible. If it is determined that an assignment is not possible, the simulator would try the second requirement in the list, and so on (the latest revision of MASS keys on a requirement, and searches down the list of aircraft for the first one that can handle the requirement).

Thi sis a test of the line break logic I want to see if it goes all the way to the right column before breaking.

A.2.2 The airlift flow problem as an optimizing simulator

The difference between a classical simulator and an optimizing simulator is in the decision function, X^{π}. There are two key differences. First, in a classical simulator, the decision function is rule-based. It uses a set of "If... THEN ... ELSE" rules to map from what we know (K_t) to a decision (which is rarely a vector). The first step in an optimizing simulator is that we require (just as an optimization model would) that we use a cost function that evaluates a decision. Let:

$C_t(x_t \mid K_t)$ be the cost of making decision x_t given our knowledge K_t.

Since the use of K_t is implicit, we will often write the cost function as $C_t(x_t)$. Furthermore, we can usually assume that the function is linear, allowing us to write $C_t(x_t) = c_t x_t$ which means that it costs twice as much if we move two aircraft as it does if we move one (with some creative modeling, we can retain our seemingly linear structure and allow the cost of moving the second aircraft to be more than the cost of moving the first). We would have to find x_t subject to basic constraints such as flow conservation (an aircraft can only do one thing at a time) and limits on flows (such as limits on how many aircraft can land at an airbase at the same time). We would express these limits using:

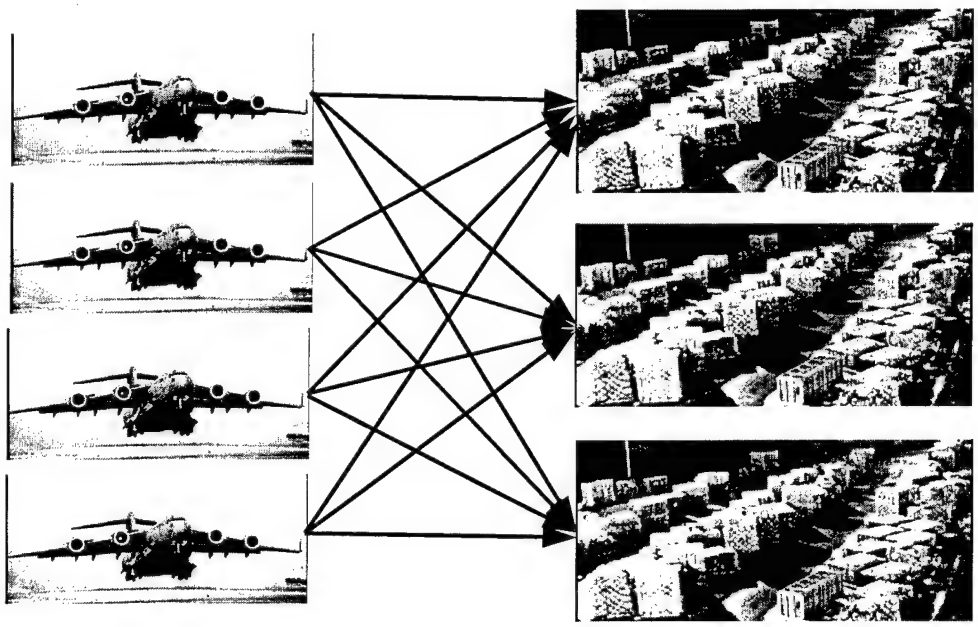

Figure A.2 – Illustration of assigning multiple aircraft to multiple requirements

$$A_t x_t = R_t$$
$$x_t \le u_t$$
$$x_t \ge 0$$

where we let:

R_t = Vector of "resources" available at time t.

For our purpose, a "resource" is anything that we are acting on that constrains the system. This would include aircraft and pilots, as well as the requirements (freight and passengers to be moved). We might also require that x_t take on integer values (to prevent us from assigning half of an aircraft to a requirement). In general, we can let:

$\mathcal{X}_t(R_t)$ = The feasible region.

We can determine what to do by solving:

$$\min_{x_t \in \mathcal{X}(R_t)} c_t x_t$$

We can now define our decision function. Recall that "arg min" means "the argument that minimizes" a function, our decision function can be written:

$$X_t^\pi(K_t) = \arg\min_{x_t \in X_t(R_t)} c_t x_t \quad (1)$$

The simplest type of optimizing simulator, then, could be written as:

$$\min_{\pi \in \Pi} \sum c_t X_t^\pi(K_t) \quad (2)$$

This is a simple myopic simulator. We require the decision function to depend purely on what we "know" at time t, represented by K_t. For the air mobility problem, we require that a cost function be developed. Solving equation (1) for the air mobility problem is very easy; it is a small network problem that can be solved using standard optimization algorithms very quickly, once the costs are calculated. Of course, this is the challenge: determining the "cost" of assigning an aircraft to a requirement requires running a small simulation to determine whether any airbase constraints are violated. We have an answer to this issue, but need to turn to it later.

It is natural to ask "Just what are we minimizing over?" That is, what is our set of policies Π over which we are choosing? This simulator is particularly simple. For a given set of costs, there is only one "policy" which is to solve equation (1).

The problem with our myopic simulator is that it does not consider the impact of decisions made now on the future. We view decisions as impacting the vector of resources in the future. To avoid adding unnecessary complexity at this stage, we are going to model decisions x_t as acting on the resource vector R_t to produce a resource vector R_{t+1} for the next time period, allowing us to write $R_{t+1}(x_t)$. Now assume that we have a function $V_{t+1}(R_{t+1}(x_t))$ which captures the *value* of having resource vector $R_{t+1}(x_t)$ at time t+1. We defer until the next section exactly how we are going to find the function $V_{t+1}(R_{t+1}(x_t))$. Using this function, we can write our decision function as:

$$X_t^\pi(K_t, V_{t+1}) = \arg\min_{x_t \in X_t(R_t)} c_t x_t + V_{t+1}(R_{t+1}(x_t)) \quad (3)$$

In our view, this is a different class of decision functions. Let Π^M be the class of *myopic* policies which have the basic form of equation (1). These policies depend purely on the knowledge K_t. If we define knowledge as data from an exogenous source, the value functions V_t represent a different class of *information* that is distinctly different than our knowledge, K_t. Rather than being provided exogenously, the value functions are calculations that we are going to undertake ourselves (endogenously).

It is useful to distinguish between *knowledge*, which is based directly on data that arrives exogenously, and *information*, which is data that is used to make a decision. We argue that the value functions V_t represent information (they are used to make a decision) but not knowledge. Since equation (3) is based on the principles of dynamic programming, we refer to policies that depend on value functions as the set Π^{DP}. Let

I_t = The set of data used to make a decision,

$X_t^\pi(I_t)$ = The decision function given the information set I_t.

We now claim that we can create different classes of policies by creating different classes of information. So far, we have seen two classes of information (K_t and V_t). Although beyond the scope of this document, we claim that there are four classes (and claim that this is all there are). For now, this is enough.

If we return to equation (2) using our new information class, we observe that the problem of choosing the best policy $\pi \in \Pi$ is the same as finding the best set of value functions $(V_t)_{t\geq 0}$. The problem now is: what should these functions look like, and how do we find them? For this, we turn to the concept of adaptive dynamic programming.

A.3 Adaptive dynamic programming

"Adaptive dynamic programming" is a name that we have given to a particular method for solving approximate dynamic programs. It falls in the general class of techniques referred to as "forward dynamic programming" which step forward in time, as opposed to classical "backward dynamic programming" methods which step backward in time. Examples of forward dynamic methods include "neuro-dynamic programming" (Bertsekas and Tsitsiklis [1996]) and "reinforcement learning" (Sutton and Barto [1998]). These approaches, however, are not able to handle the types of operational problems that arise generally in resource management, and specifically in the area of transportation and logistics.

We begin our presentation with a review of forward dynamic programming techniques. We then discuss the "three curses of dimensionality" that arise when solving problems in transportation and logistics. Finally, we present a novel approximation strategy that overcomes the "three curses."

A.3.1 An introduction to forward dynamic programming

The first step in our solution approach is to develop a decision function that captures the impact in the future of decisions made at time t. Normally this could be accomplished using the optimality equations of dynamic programming. Assume that we have a system in state S_t and let $V_t(S_t)$ be the value of being in state S_t. Under the assumption that the state variable captures all relevant history of our process, the value functions are determined by:

$$V_t(S_t) = \min_{x_t \in X_t(S_t)} c_t x_t + E\{V_{t+1}(S_{t+1}(x_t)) \mid S_t\} \quad (4)$$

Equation (4) can in principle be solved using backward dynamic programming. Assume that $V_T(S_T)$ is known for a terminal period T. We can then find all remaining functions

$V_t(S_t)$ by stepping backward through time using equation (4). This process requires solving equation (4) for each possible value of S_t (the assumption is that S_t is discrete). The problem with this approach is that if S_t is a vector, then we encounter the well-known curse of dimensionality: there are simply too many states.

In recent years, considerable attention has been given to the techniques of forward dynamic programming. These have been popularized in the textbooks by Bertsekas and Tsitsiklis [1996], under the label of "neuro-dynamic programming," and Sutton and Barto [1998], under the label of "reinforcement learning." These techniques mitigate, but do not eliminate, the curse of dimensionality of the state space by solving equation (4) by stepping forward through time using an approximate value function. Let $\hat{V}_t^n(S_t)$ be an estimate of the value of being in state S_t at iteration n. Then, the basic forward dynamic programming algorithm works as follows:

Step 0: Initialize $\hat{V}_t^0(S_t)$, and pick an initial state S_0.

Set the iteration counter $n = 0$.

Set the maximum number of iterations N.

Step 1: Set $S_0^n = S_0$, and select $\omega^n \in \Omega$.

Step 2: (Forward pass)

Do for $t = 0,1,...,T-1$:

Step 2.1 Let $\omega_t^n = \kappa_t(\omega^n)$.

Step 2.2: Solve:

$$\tilde{V}_t^{n+1}(S_t^n, \omega_t^n) = \min_{x_t \in X_t(S_t)} c_t(\omega_t^n)x_t + E\left\{\hat{V}_{t+1}^n(S_{t+1}(x_t), \omega_t^n) \mid S_t^n\right\} \qquad (5)$$

Let x_t^n be the optimal solution of equation (5).

Step 2.3: Update the state of our system using:

$$S_{t+1}^n = f_t(S_t^n, x_t^n, \omega_t^n)$$

where $f_t(S, x, \omega)$ is the *transfer function* describing the dynamics of our system.

Step 3: (Backward pass):

Do for $t = T-1, T-2, ..., 0$:

Update the value functions:

$$\hat{V}_t^{n+1} \leftarrow U^V(\hat{V}_t^n, \tilde{V}_t^{n+1}, S_t^n)$$

Step 4: If $n < N$, set $n \leftarrow n+1$ and return to **Step 1.**

Our presentation of the forward dynamic programming algorithm has been intentionally stated in the context of a finite horizon problem, since this is how the air mobility problem (and every other resource management problem that we have ever faced) needs to be formulated. Note that the textbooks by both Bertsekas and Tsitsiklis as well as the Sutton and Barto are presented in the context of steady state problems. The transition to finite horizon problems is straightforward but nontrivial, and create practical problems that are not addressed in these presentations.

A.3.2 The three curses of dimensionality in resource management

For resource allocation problems that arise in operational problems such as the airlift mobility problem, the problem with equation (4) is that there is not one curse of dimensionality, but three! These are:

1. The state space – If S_t is a vector, then we do indeed suffer from the problem of having to determine $V_t(S_t)$ for each element of an exponentially large state space.

2. The outcome space – Random variables (future demands, weather delays, equipment breakdowns) typically arise in vectors, sometimes of large dimensionality. The expectation operator in (4) requires summing over all possible values of the random vector. As with the state space, the number of potential outcomes is exponentially large.

3. The action space – Operational problems are characterized by "actions" that come in vectors, as illustrated in figure A.2. In backward dynamic programming, it is expected that equation (4) will be solved by computing the argument for each possible value of x_t in the feasible region $\mathsf{X}(S_t)$. Again, if x_t is a vector, then the size of $\mathsf{X}(S_t)$ becomes exponentially large.

The dynamic programming literature has tended to focus on the problem of large state spaces; we have found the biggest difficulty is in the outcome and action spaces. We eliminate the problem of the outcome space by reformulating the optimality recursion around a concept we refer to as the *incomplete state variable*. We solve the problem of the large state spaces by using continuous functional approximations. We solve the problem of large action spaces by choosing the structure of our functional approximations carefully.

A.3.3 Solving the three curses of dimensionality

We solve the three curses of dimensionality using two key strategies. First, we formulate our optimality equations using the concept of an incomplete state variable. Then, we replace the value function with a suitably chosen continuous approximation that helps us to retain the structure of the original myopic decision problem.

A.3.3.1 The incomplete state variable

Earlier, we represented the sequence of exogenous and endogenous information as the sequence:

$$(K_0, x_0, \kappa_1, x_1, \kappa_2, x_2, \ldots)$$

When modeling dynamic systems, it is common to represent the concept of the "state" of a system. Often overlooked is that there are three concepts of the "state" of a system: the state of our knowledge, the state of the resources, and the state of a single resource in the system. To avoid any confusion, we let K_t represent the state of our knowledge, and we let:

$$\begin{aligned} R_{at} &= \text{The number of resources with attribute } a \in \mathsf{A} \\ R_t &= \text{The state of our resources} \\ &= \left(R_{at}\right)_{a \in \mathsf{A}} \end{aligned}$$

The vector R_t is our system resource state vector, while $a \in \mathsf{A}$ is the attribute (state) of a single resource. We find it is useful to define these concepts because different communities will use the word "state" to mean any one of these three concepts. We use the variable S_t when we wish to refer to the state in a generic way, without distinguishing whether we are referring to the more general concept of knowledge, or the more specific concept of resources.

It is customary to represent the state of the system as consisting of what we know immediately before we make a decision. In this case, we would have:

$$(K_0, x_0, \kappa_1, S_1, x_1, \kappa_2, S_2, x_2, \ldots)$$

This definition leads to the classical optimality equations given by:

$$V_t^{n+1}(S_t) = \min_{x_t \in X_t(S_t)} c_t x_t + E\left\{V_{t+1}(S_{t+1}(x_t)) \mid S_t\right\}$$

We have found it more useful to refer to this version of a state variable as the *complete* state variable, which we denote by S_t^+. We then use the simpler S_t to denote the *incomplete* state variable that is the information we know just before the information arrives in time t. This implies that our sequence of information and states would be represented by:

$$(K_0, x_0, S_1, \kappa_1, S_1^+, x_1, S_2, \kappa_2, S_2^+, x_2, \ldots)$$

In this case, the optimality equations (formed using the incomplete state variable S_t) are given by:

$$V_t(S_t) = E\left\{ \min_{x_t \in X_t(S_t)} c_t x_t + V_{t+1}(S_{t+1}(x_t)) \mid S_t \right\}$$

Note that the expectation operator is now in front of the optimization operator. The reason is that our state variable no longer has all the information we need to make a decision.

A.3.3.2 Approximating the value function

Our next step is to simply drop the expectation operator, and solve the problem for a *single* sample realization:

$$V_t(S_t, \omega_t) = \min_{x_t \in X_t(S_t, \omega)} c_t x_t(\omega_t) + V_{t+1}(S_{t+1}(x_t, \omega_t))$$

Notice that we are solving the problem for a single realization of the exogenous information in time period t, represented by $\omega_t = \kappa_t(\omega)$. Had we used our complete state variable and used the same trick, we would have to solve the problem for a single realization of $\omega_{t+1} = \kappa_{t+1}(\omega)$. This means that we are making a decision in time t assuming that we know what would happen in time t+1.

The second step is to replace the value function $V_{t+1}(S_{t+1})$ with an approximation. We specifically require that our approximation be a continuous function of the *resource* state vector, R_{t+1}. Thus, we replace $V_{t+1}(S_{t+1})$ with $\hat{V}_{t+1}(R_{t+1})$, giving us the equation:

$$\tilde{V}_t^{n+1}(S_t, \omega_t) = \min_{x_t \in X_t(S_t, \omega)} c_t x_t(\omega_t) + \hat{V}_{t+1}^n(R_{t+1}(x_t, \omega_t))$$

The last step is to choose reasonable functional forms for the approximation $\hat{V}_{t+1}(R_{t+1})$. We have been experimenting with two specific forms:

$$\hat{V}_{t+1}(R_{t+1}) = \sum_{a \in A} \hat{v}_{a,t+1} R_{a,t+1} \tag{6}$$

$$\hat{V}_{t+1}(R_{t+1}) = \sum_{a \in A} \hat{V}_{a,t+1}\left(R_{a,t+1}\right) \tag{7}$$

The first approximation is linear, while the second is nonlinear, separable. In particular, we have used piecewise linear, separable functions. Our work has shown that linear approximations are the easiest, but do not produce the best results. Separable, nonlinear

(piecewise linear) appears to be providing exceptionally good results. While nonlinear approximations do introduce added complexity, the stability of these solutions and the quality is quite good.

A.3.3.3 Exploiting structure

Solving the equation:

$$\tilde{V}_t^{n+1}(S_t,\omega_t) = \min_{x_t \in X_t(S_t,\omega)} c_t x_t(\omega_t) + \hat{V}_{t+1}^n(R_{t+1}(x_t,\omega_t))$$

generally requires exploiting underlying problem structure. The basic myopic problem:

$$\tilde{V}_t^{n+1}(S_t,\omega_t) = \min_{x_t \in X_t(S_t,\omega)} c_t x_t(\omega_t)$$

can be a sort, an assignment problem (such as assigning aircraft to requirements), a transportation problem, an integer multicommodity flow problem, or a discrete routing and scheduling problem. We make the assumption that we have an algorithm to solve this problem, which is typically not too difficult because it is only for one point in time (solving problems simultaneously over an entire planning horizon can be exceptionally difficult; solving only one time period is often quite easy). When designing a functional approximation $\hat{V}_{t+1}^n(R_{t+1}(x_t,\omega_t))$, the goal is to choose a form that does not destroy the inherent structure of the one-period problem. Linear approximations of the form (6) never destroy problem structure. Separable nonlinear approximations (equation (7)) work very well for single commodity resource allocation problems since they retain network structure. In the context of multicommodity flow problems, linear approximations will produce pure network subproblems, while nonlinear approximations produce multicommodity flow problems (which are relatively small).

A.4 The dynamic resource transformation problem

Our work on optimizing simulators has paralleled the formulation of a very general problem class called a *dynamic resource transformation problem.* This problem class represents an effort to identify the major elements of a resource management problem, and to provide a notational system that elegantly handles the complexities of real-world operations. A detailed summary of this problem class is given in reference [1].

A.4.1 Elements of a DRTP

A "DRTP" consists of three primary dimensions:

Resources || Processes || Controls

Each of the three primary dimensions consists of specific subdimensions. A complete notational system is provided. The representation introduces several novel modeling elements:

- The introduction of an attribute vector instead of the more common "multicommodity flow" formulation that is popular in the literature). This gives way to the heterogeneous resource allocation problem which is a much more flexible notational system for handling complex resources. This dramatically simplifies modeling problems of increasing complexity without adding notational complexity.

- The concept of resource layering, whereby resources may be coupled (such as aircraft with pilot) to produce resources with a richer set of attributes.

- The "modify" function which is a single function which captures *all* of the physics of a problem Similar to the transfer function of simulation, the modify function more naturally works at the level of individual resources.

- Explicit modeling of not only the flow of information (common to any simulator) but also the organization of information and decisions.

- Formalization of four classes of information, which produces four classes of optimization algorithms.

- Introduction of the use of hierarchical aggregation for dynamic resource management problems. Aggregation brings into play the explicit representation of information, a dimension that is typically overlooked in modeling, which typically works at a single level of aggregation. DRTP's will in general use three levels of aggregation at the same time (although more levels can in principle be used).

In addition, we have introduced an *algorithmic metastrategy* for solving general DRTP's. This metastrategy is based on the decomposition of problems based on *information subproblems*, and the introduction of a novel communication strategy based on the use of nonlinear CAVE approximations.

A.4.2 Major problem classes

Resource allocation problems come in a variety of flavors. Important dimensions include:

1. The information profile – Our work has primarily focused on problems where there is limited advance information. When all information is known in advance, we typically face very large-scale optimization problems. Many real problems (which encompasses the airlift problem as well as most freight transportation problems) involve information that arrives over time.

2. Resource attributes – Resources may have a single static attribute (such as product class), a single dynamic attribute (such as the location of an aircraft), both a static and dynamic attribute (different types of aircraft spread across

different locations), or multiple attributes (the maintenance or fuel status of the aircraft, what freight it is carrying, and the characteristics of the crew that is piloting the aircraft).

3. Resource layers – The simplest problems are single layer problems (moving aircraft around or delivering products to customers); in transportation, many problems can be reasonably solved as two layer problems (such as aircraft and requirements, or pilots and flights). Harder problems have multiple layers.

Our work has focused on problems with highly dynamic information processes, and relatively complex attributes (we have found that it is most natural to simply view resources as having a vector of attributes, rather than starting with the so-called simpler single and multiple-commodity flow problems). However, we have started with single and two-layer problems, and have found that this has been a significant issue that is worth a brief discussion.

A.4.3 Resource layering

For the airlift problem, aircraft and requirements are both resource classes, and since we have to model an aircraft when it is moving a requirement, we have to model the resource *layer* of aircraft with requirement. Requirements are a passive layer (we cannot move a requirement without an aircraft), so this is approached as a one-layer problem. If we could move requirements on aircraft not controlled by the AMC (e.g. charter aircraft), then this problem would have two active layers. If we wanted to simultaneously model crews, we would effectively have a three layer problem.

If a problem has a single resource class that we are managing (such as aircraft or pilots) and they have to perform tasks that *must* be performed at a particular point in time, then we have a pure single layer problem. If both layers are active (we can manage the aircraft, but we also have to perform functions with the requirements), then we have two active layers, and this is a true two-layer problem. Many problems involve a single active layer (such as the aircraft) and a single passive layer (you can act on a requirement by moving it with an aircraft, and if you do not move it, it just sits there, but you cannot directly act on the attributes of the requirement without the aircraft). The requirements are a passive layer because you cannot act on them, but if you do not move them, then they are still there the next time period. These problems arise frequently in practice, and they can be modeled as one or two layer problems (depending on how precise you want to be).

A.4.3.1 Single-layer problems:

A single layer problem has a single resource class that we are managing. Single layer problems arise when we are managing aircraft to move requirements, scheduling pilots to handle flights, or moving trucks to move loads. These problems come in three important flavors:

1. Single commodity problems
2. Multicommodity problems
3. Heterogeneous resource allocation problems

Single commodity problems involve a single type of resource (for example, all the aircraft are the same), which are characterized only by a state variable (such as their location). Multicommodity problems are characterized by both a state (location) and a class (type of aircraft). Heterogeneous resource allocation problems are characterized by a vector of attributes, some of which may be static while others are dynamic. We have found that this is an especially important class. The complication here is that the space of potential attributes for complex resources can be extremely large (especially those involving people or complex equipment such as aircraft or locomotives). Special strategies have to be formulated to handle this problem.

A.4.3.2 Two-layer problems:

Two layer problems first arise in problems such as the airlift mobility problem where there is a single active resource class (the aircraft) and a single passive resource class (the requirements). Imagine now that it may be necessary to move freight from one location to the next, unload it, and then reload it on another plane to be moved over a subsequent leg. Furthermore, part of our problem is determining which sequence of airbases a requirement should be moved through to get to the final destination.

In this setting, we have to manage both the aircraft and the requirements. This is a common example of a two-layer problem. An added complication arises when the requirements vary not only in terms of their physical characteristics (which affects which aircraft is best to move the freight) but also in terms of importance. As a result, it may be necessary to leave a low priority requirement on the ground while an aircraft waits for a higher priority requirement to arrive before it can be moved. To capture this behavior, the model needs to think about not only the future impact of moving an aircraft, but also the future impact of moving a requirement. For example, it might be possible for a requirement to get to a destination by moving through airbases 1-2-3 (where the freight is unloaded at 2), but if 2 is heavily congested, it might be better to move the requirement over the route 1-4-3, taking advantage of available capacity at 4.

We have studied three classes of two-layer problems:

- Resource allocation problems serving tasks with time windows (but where a task vanishes after being moved)
- Combined fleet management and traffic assignment problems (where customer demands have to be routed through multiple locations before arriving at the destination).
- The dynamic assignment problem

The dynamic assignment problem arises when you have a set of resources (such as pilots) and a set of tasks (such as flights to be served) and we have to dynamically assign

resources to tasks. Further assume that both resources and tasks have potentially complex sets of attributes that affect the value of assigning a resource to a task).

A.4.3.3 Multi-layer problems

Many real problems come not in one or two layers, but three or more. For the airlift problem, an example is the modeling of aircraft, crews and freight. We could create a fourth layer by adding fuel.

We have solved a five layer problem (driver, tractor, trailer, chemical product, and customer tank) for a chemical manufacturer, Air Products and Chemicals. The problem was solved in a general way, and the algorithm, in principle, could be adapted to other multilayer problems. This work has not yet been formally documented.

A.5 Experimental results

We have tested this strategy on a broad range of deterministic and stochastic datasets that we use in CASTLE Laboratory. The ideas have also been tested on real-world problems such as the driver management problem at Yellow Freight, and the locomotive management problems at Norfolk Southern and Burlington Northern Sante Fe railroads. In this section, we focus purely on the laboratory datasets, since the real world datasets bring other dimensions of our technology into play.

A.5.1 Convergence of the CAVE algorithm

Much of our work uses an important foundational algorithm called the CAVE algorithm, which uses a sequence of sample gradients of a function to build up a piecewise linear, concave approximation. This year, we finally proved that this algorithm is convergent.

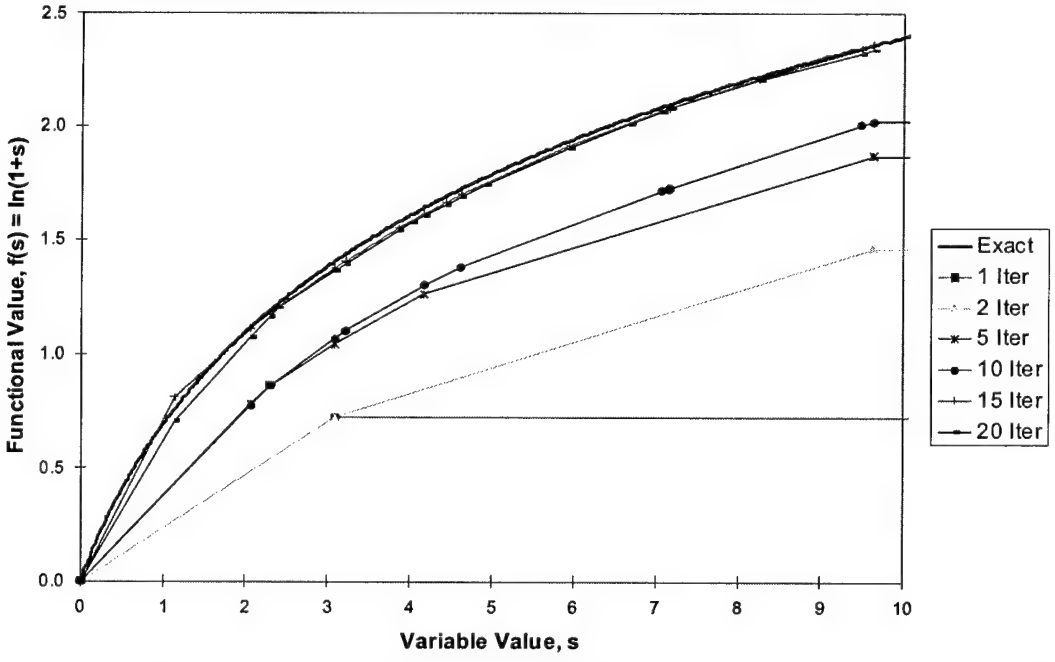

Figure A.3 – Convergence of the CAVE algorithm to a nonlinear function using a series of sample gradients.

Figure A.3 demonstrates the progress of the algorithm, starting with a single line, and iteratively improving the approximation of a nonlinear function.

We now have a theoretical proof of convergence of the CAVE algorithm (the report is in preparation). While this represents an important milestone (which proved to be surprisingly difficult), of greater practical importance is the fast rate of convergence exhibited by the algorithm.

A.5.2 CAVE on two-stage allocation problems with no substitution

We use the CAVE algorithm to produce separable, piecewise linear approximations of the value of resources in the future. This is one of our most powerful tools for solving resource allocation problems. We illustrate our technique on a series of problems of increasing complexity. Figure A.4 is the simplest example. Here we illustrate the assignment of four different types of aircraft to different locations, after which they have to be assigned to demands (that are initially unknown). In this version of the problem we assume that once we move a resource (aircraft) to a location, that it will then be assigned to demands that can only be served from that location. This is a very simple stochastic resource allocation problem that we can solve to optimality, providing a nice benchmark for comparison. Our optimal solution, however, only works for this special case.

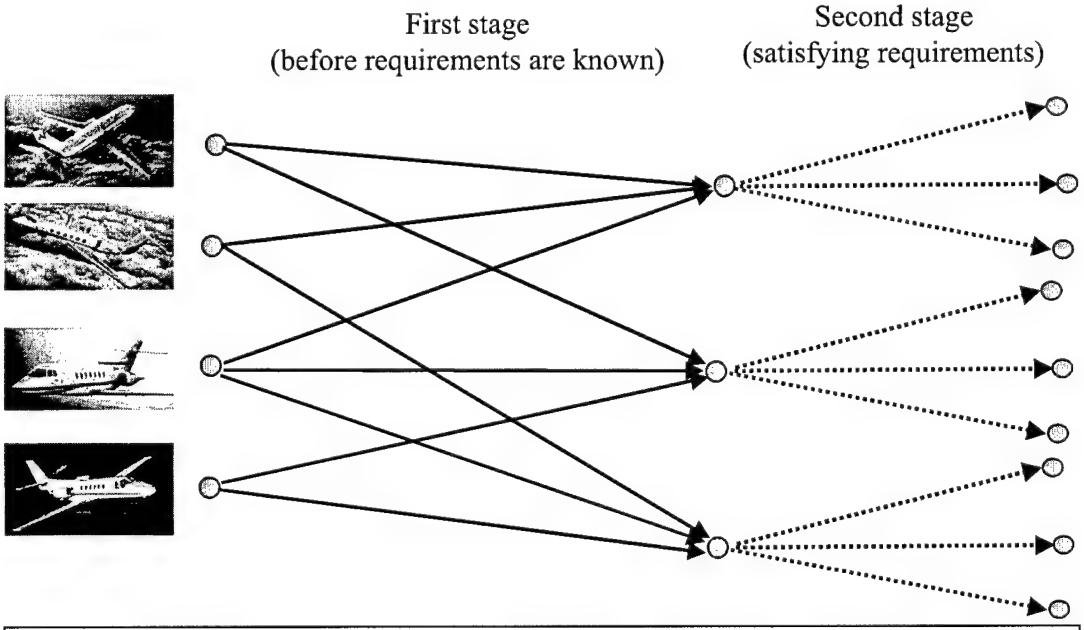

Figure A.4 – Assigning aircraft now with an uncertain but separable future.

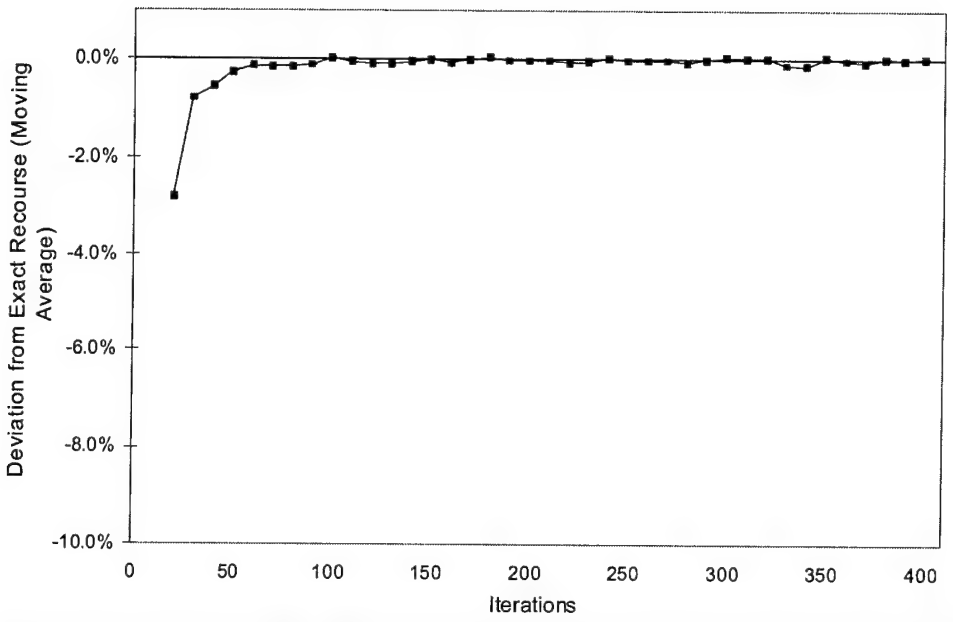

Figure A.5 – Convergence of the CAVE algorithm to the optimal solution.

Figure A.5 shows the convergence of the algorithm to the optimal solution. Of particular interest is the fast rate of convergence, and the stability of the solution. This is especially important for large scale problems (where we simply cannot run that many iterations),

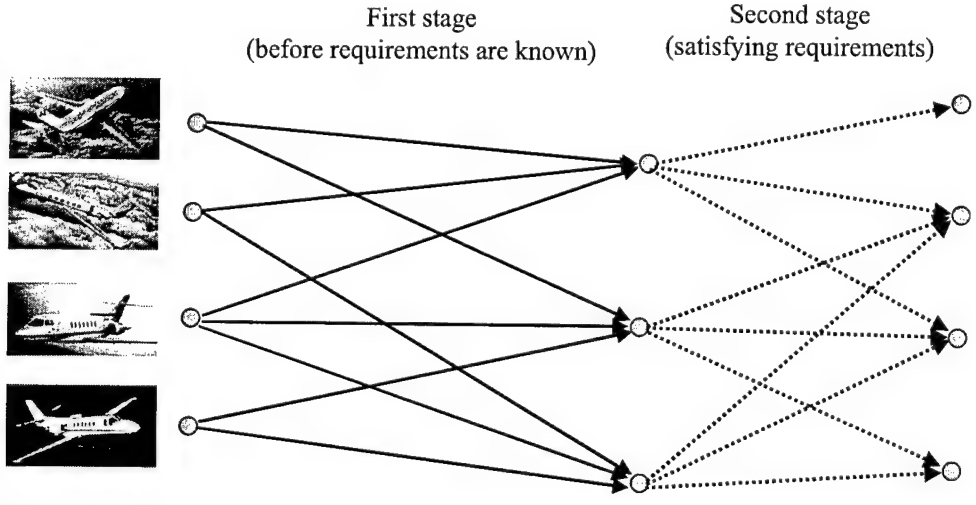

Figure A.6 – We may allocate aircraft under uncertainty, but where there is substitution in the future. This will create a nonseparable function of future rewards.

and the stability is important when analyzing "what if" scenarios.

A.5.3 CAVE on two-stage allocation problems with substitution

The next problem class is similar to the first, but now we add the important element of realism where after moving our aircraft, we "see" the requirements and then we can assign aircraft to requirements with substitution. Figure A.6 illustrates a problem with substitution. Here, we have to allocate aircraft before we know exactly how they are needed. Afterwards, we are allowed to make substitutions between aircraft.

Figure A.7 below shows comparisons between the CAVE algorithm (which is not provably optimal for this more general problem) against the CUPPS algorithm which is provably optimal. CUPPS is in the family of nested-Benders algorithms which use cutting planes derived from the dual solution of the second stage problem. Although CUPPS is optimal, we found that the CAVE algorithm produces almost identical solutions for smaller problems. By contrast, CAVE actually outperforms CUPPS for larger problems because it convergences much more quickly.

A.5.4 The CAVE algorithm on multicommodity flow problems

The next experiment applies the CAVE algorithm to a deterministic, integer, multicommodity flow problem (which describes the version of the airlift mobility problem that is solved when it is formulated as an optimization problem). We compared the CAVE algorithm to the solution produced by a commercial LP solver (which does not

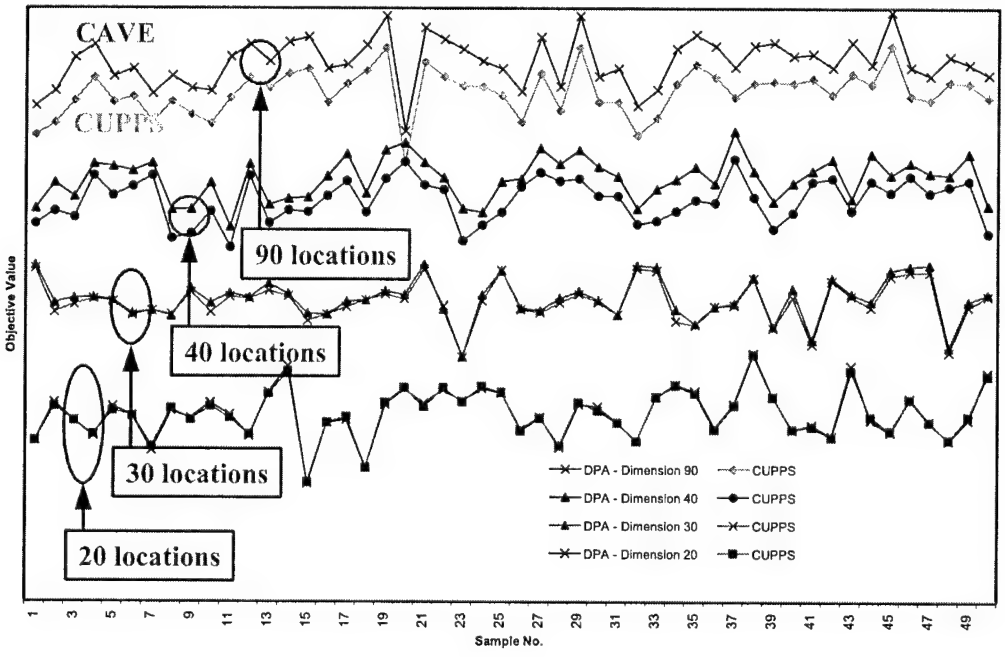

Figure A.7 – Comparison of the CAVE algorithm against an optimal solution from Benders after 200 learning iterations. As the number of locations increases, the CAVE approximations outperforms Benders because of faster convergence.

provide integer solutions). The results are shown in figure A.8, which show that the

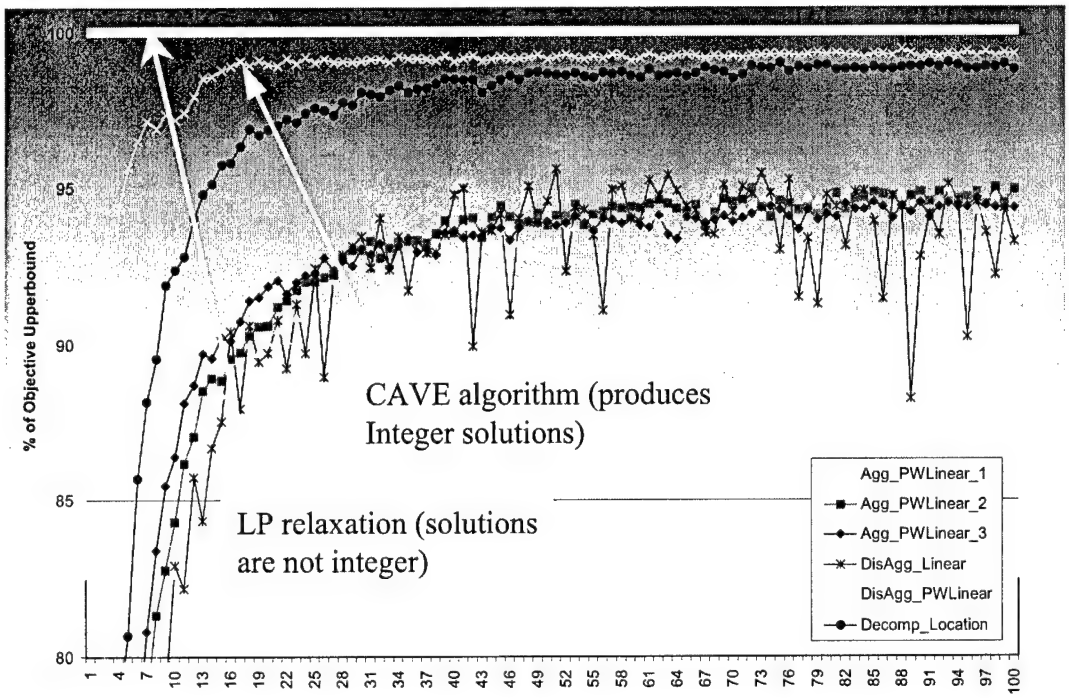

Figure A.8 – The adaptive estimation logic produces near-optimal solutions on deterministic datasets which are also integer. Here we are within half of a percent of the optimal solution produced by a commercial solver (which does not return integer solutions).

CAVE algorithm is producing very near optimal solutions for this very difficult problem class.

A.5.5 The CAVE algorithm in stochastic simulations

Our last experiment reports on the results of a series of simulations comparing the performance of simulating decisions solved using the CAVE approximations, against those obtained by solving rolling horizon problems using (point) forecasts of future events. We note that once the value functions have been estimated, the simulation performed using CAVE is of comparable complexity to a traditional simulation. However, our method normally will require running 20-30 iterations to properly warm up the nonlinear approximations.

Figure A.9 shows the difference between making decisions based on point forecasts versus using nonlinear functions estimated using sample estimation. The difference is significant. We note that these results are for problems where a customer demand must be served in the time period in which it becomes known. Other problems (which include

the airlift mobility problem) allow requirements to be served after they first become available.

A.5.6 The dynamic assignment problem

The dynamic assignment problem involves assigning resources to tasks over time. Both resources and tasks may arrive in the future. Also, the cost of assigning a resource to a task may be uncertain. For example, we may not be able to perfectly predict the "cost" of assigning a particular aircraft to a requirement because of the difficulty of predicting airbase congestion or equipment failures in the future.

Dynamic routing and scheduling problems are most often solved, both in the research

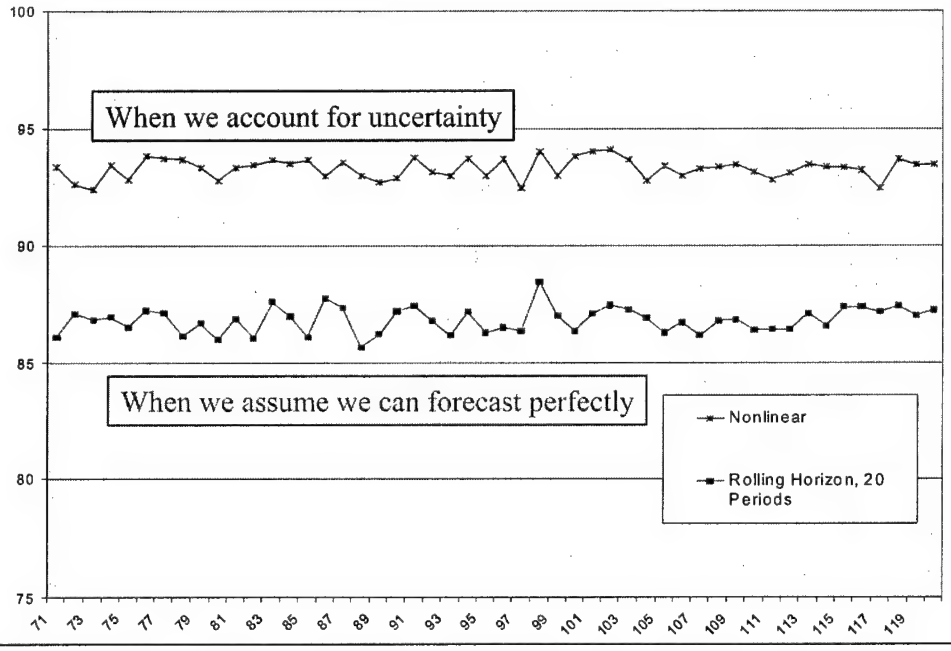

Figure A.9 – Rolling horizon procedures use a point forecast of the future, which means we plan for a specific outcome. Our methods produce more robust solutions which work better as the future evolves.

literature and in engineering practice, using myopic models. The problem is solved at time t using only the information known at time t. A major weakness of these algorithms is that it can produce results where a particular resource is assigned to a task even though the assignment is a very poor one, and any reasonably experienced operations specialist would know to hold off on the assignment and wait for better options.

The dynamic assignment problem is quite different from the other resource allocation problems we have considered because the problem is highly discrete (all decisions are 0/1) and the attribute space is extremely large. We solved the problem using adaptive dynamic programming techniques, comparing four classes of approximations: zero (that

is, ignoring the impact of decision on the future), resource gradients (we consider the value of a resource in the future), resource and task gradients (we consider the value of both resources and tasks in the future), and "arc gradients", where we consider the value of a resource/task *pair* in the future (effectively a nonseparable approximations). All of these approximations involve solving sequences of simple assignment problems, so the computational complexity is no higher than a simple myopic model, with the exception that we have to simulate the problem repeatedly in order to estimate the value functions.

The results using deterministic data indicate that we can obtain results that are very close to the optimal solution when we use the "arc gradients" approximation. This particular approximation is the most computationally demanding. Interestingly, when we use stochastic data, we get the best results when using both resource and task gradients, which are computationally fairly easy to obtain.

The dynamic assignment problem is characterized by a large attribute space. It is natural to ask whether aggregation can be used effectively to help improve the value function. For example, we might divide a region geographically into a 10x10 grid. Needless to say, it will take a large number of observations to obtain a statistically valid estimate of the value of a resource in all 100 grid squares. Fewer iterations would be needed if we used a 5x5 grid pattern, but we would introduce larger structural errors. The problem is the classic tradeoff in statistical model fitting; if we use too many variables, we introduce statistical error, while if we use too few variables, we introduce structural error.

Figure A.10 shows the results of five sets of experiments, using value functions at five different levels of aggregation. The most aggregate representation produces the best

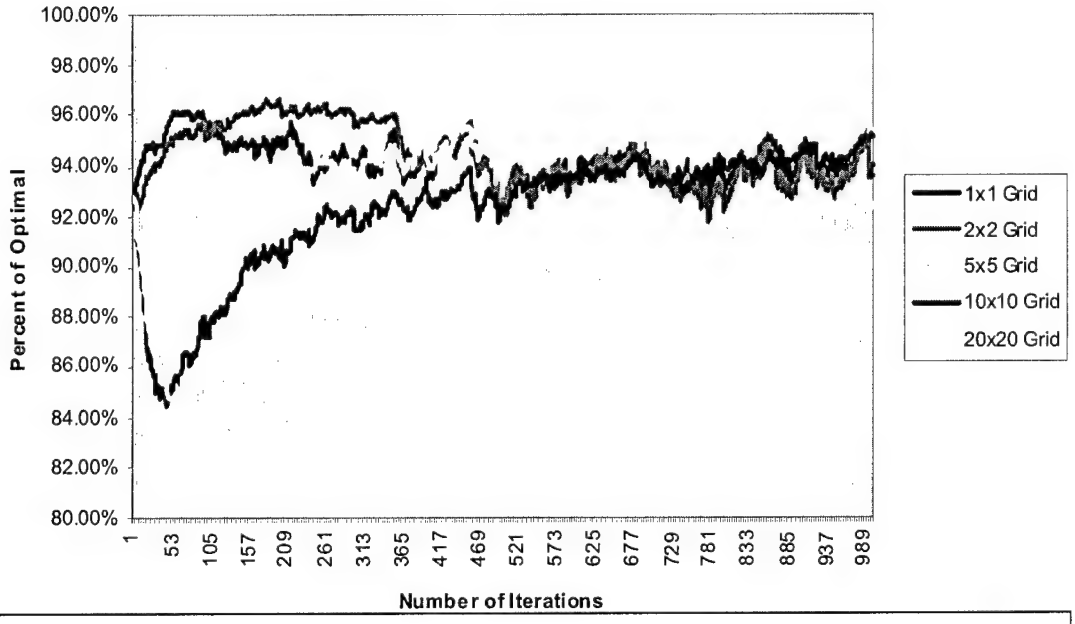

Figure A.10 – More aggregate representations perform best with fewer iterations; more detailed (less aggregate) representations work better in the limit.

results when we use the smallest number of iterations. The second most aggregate model (where the geographical region was broken into a 2x2 grid) produced the best results for the next set of iterations (between approximately 100 and almost 400 iterations). The more disaggregate representation we used, the more iterations that were needed to produce the best results.

These experiments suggested that a hybrid scheme might work best. Here, we compute value functions at all levels of aggregation, and we use the estimate with the smallest variance for each resource and task. We note that this logic might use a more disaggregate estimate for resources and tasks in locations where there has been a lot of activity, and more aggregate estimates in areas where there have been fewer observations. Figure A.11 shows the results of a test of a hybrid strategy, which indicates that it

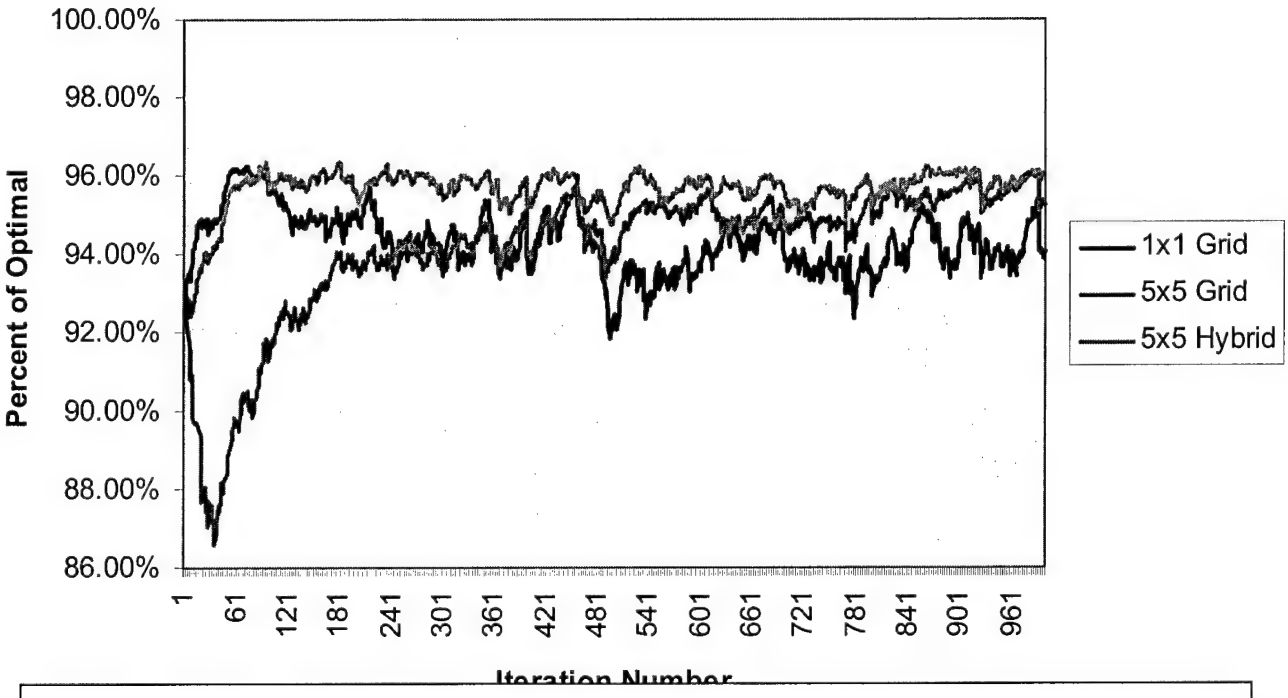

Figure A.11 – If we used a hybrid scheme combining different levels of aggregation at the same time, we get good results over the entire range of the algorithm.

produces consistently good results over the entire range of the algorithm.

A.6 The PILOTVIEW diagnostic system

We have found that when we model complex systems, we often do not understand "why the model did that." Furthermore, we also typically find that when we theorize why the model behaves in a certain (usually undesirable way), that we are usually wrong.

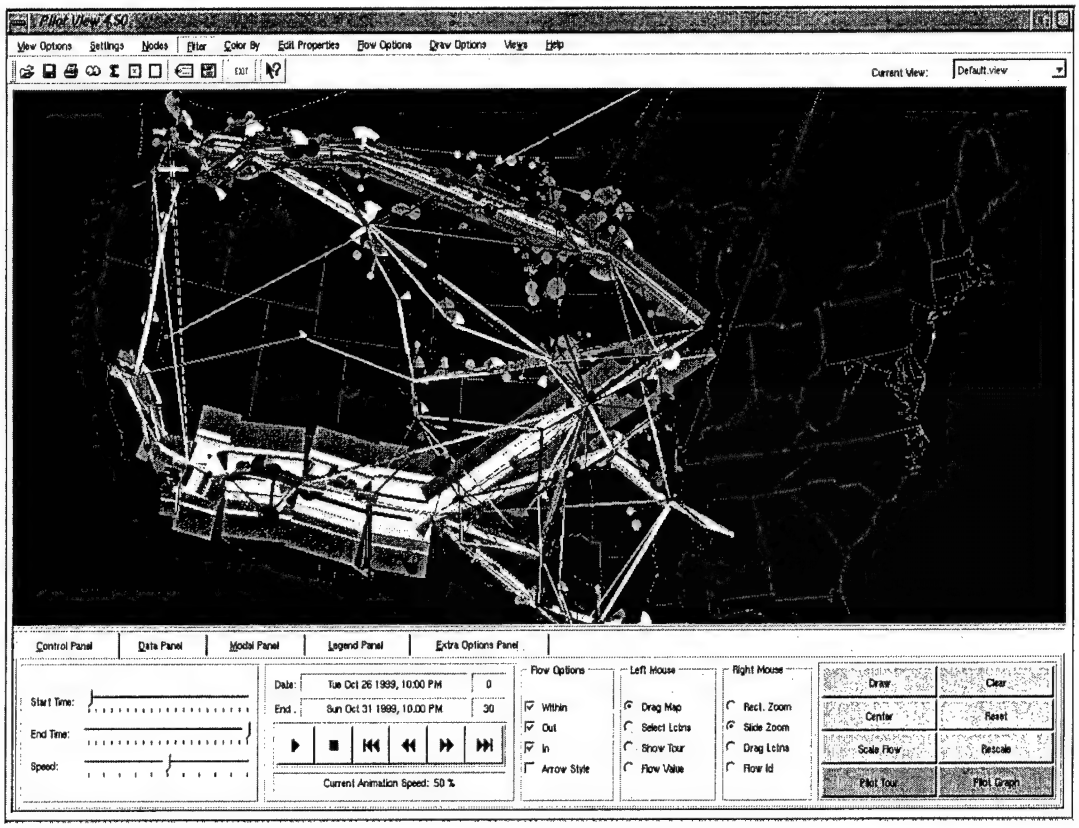

Figure A.12 – A Snapshot of flows moving around the country, colored by an attribute.

Simulations can provide errant results because of one of four reasons: 1) the data is wrong, 2) the model is wrong (for example, the costs of decisions), 3) there is a flaw in the algorithm, or 4) there is a bug in the software. The problem is that identifying which of these is the culprit for a particular behavior is very difficult.

We have developed a general purpose diagnostic system called PILOTVIEW. This system is designed to work on general, multi-layer, multiattribute dynamic resource transformation problems. As a result, we are able to apply the system to problems involving aircraft, trucks or trains. We can model two-layer problems (aircraft and requirements; drivers and loads; locomotives and trains) or multilayer problems (at one company, we are modeling five layers: driver, tractor, trailer, product and customer).

PILOTVIEW operates on two types of datasets. The first is an activity file that gives the flows of resources (of different types, with different attributes) over time. There are three basic views for this data file: a static view (plots of flows over an interval of time –A.12),

an animation of activities (showing objects moving over a map over time – Figure A.13), and a graph of activities plotted over the entire horizon of the simulation.

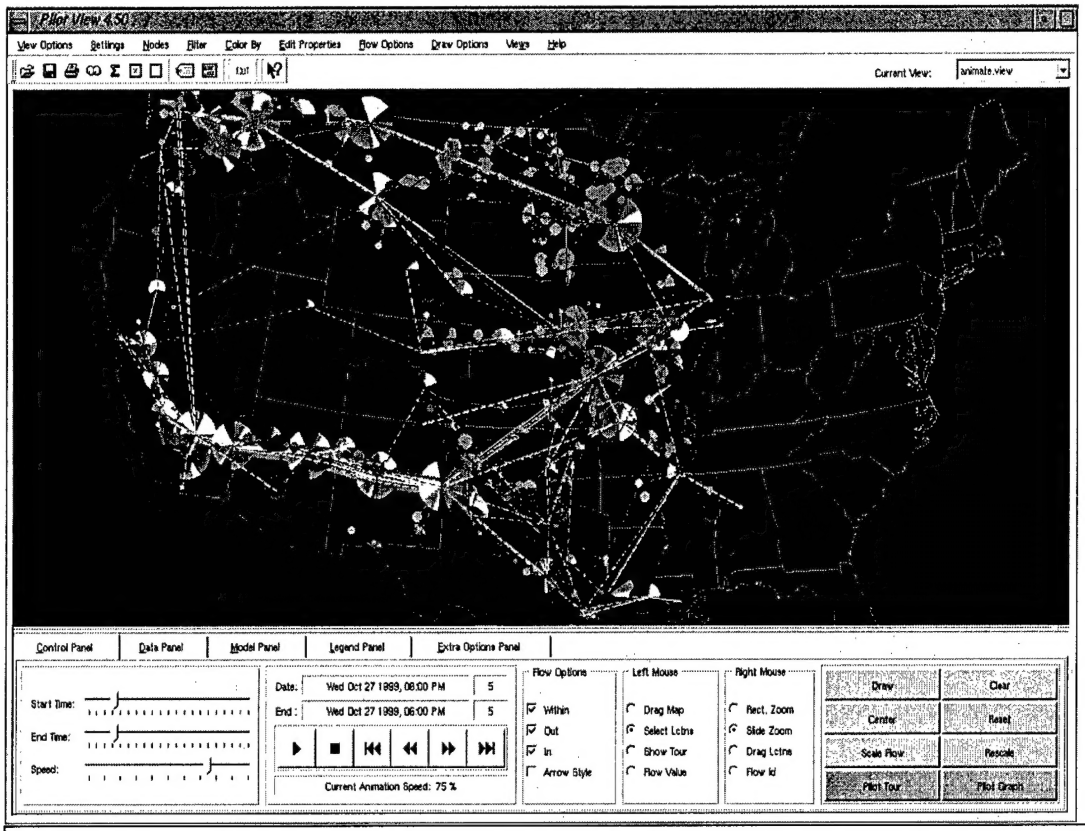

Figure A.13 – A snapshot of an animation, which shows activities moving from one location to the next.

Perhaps the most powerful tool within PILOTVIEW is the "Pilottour" module, which brings us right inside the optimization model itself. This is the tool that allows us to understand why the model made a particular decision. Pilottour works with a representation that a "resource" will have a vector of attributes "a", where there may be more than one resource with the same set of attributes. Pilottour displays "informational subproblems" which represent blocks of information that make up a single subproblem. Geographically, an informational subproblem can be the entire system (at one point in time), a single location, or a geographical area. Figure A.14 illustrates a number of different subproblems, each shown as a box with two columns of boxes (each column representing a resource layer). For the airlift problem, the first column represents types of aircraft, while the second column is requirements). Pilottour allows the analyst to click on any box or line (representing a possible decision) to obtain drill-down information. We have found that this tool allows people other than the original programmer of the model to analyze and diagnose problems.

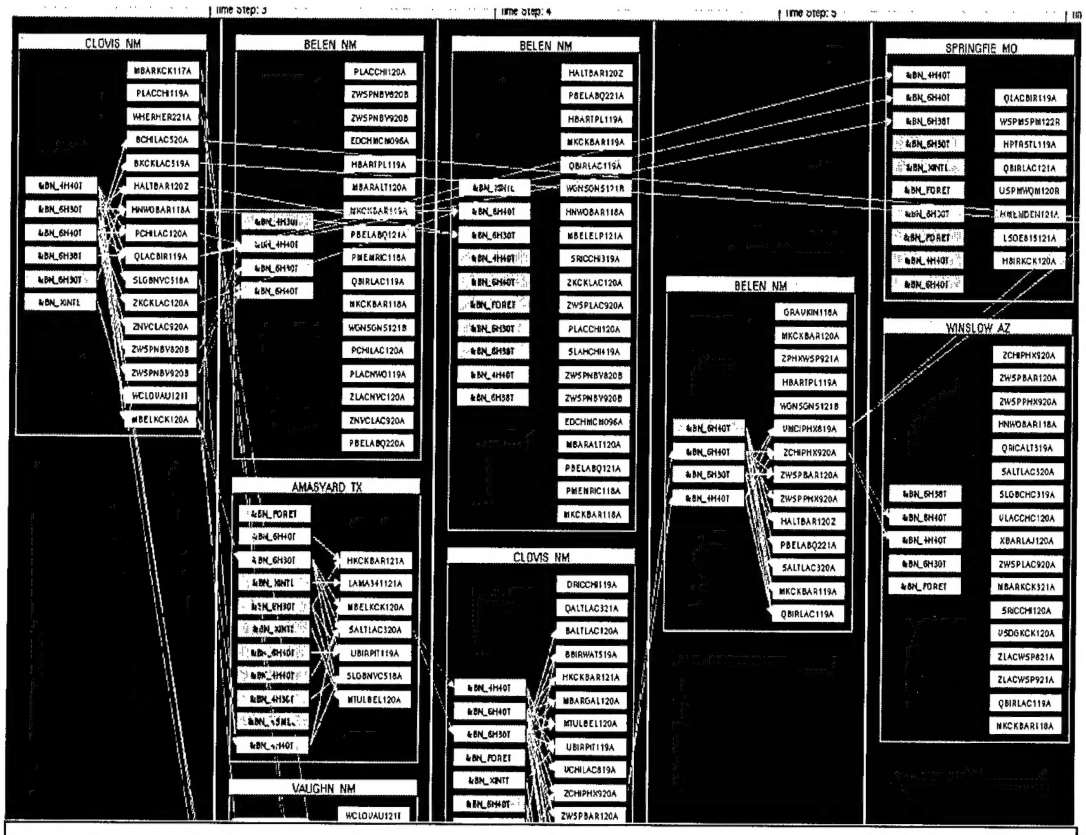

Figure A.14 – PILOTTOUR allows the user to go inside an optimization model, looking not only at the decisions that were made, but also the decisions that were not made, and why.

A.7 Optimization with patterns: combining "OR" and "AI"

One of the most powerful techniques that we have learned represents a merger of "OR" and "AI". "OR" typically refers to minimizing a cost function to produce a set of decisions. "AI" "AI", by contrast, uses set of rules, which represent a mapping of a state to a decision. The limitation of AI techniques is that for general resource allocation problems, the state of the system is hopelessly complex, making the design of general sets of rules very cumbersome. This limitation usually forces modelers to simplify the state of the system, but this in turn can produce undesirable behaviors. The challenge is striking a balance between the complexity of the rules and the quality of the solution.

Optimization methods ("OR") on the other hand, depend on the definition of a cost function to get the desired behavior. If an accurate cost function can be specified, and if an algorithm can be designed to minimize the cost function, the payoff is a very general and flexible system that responds nicely to different datasets. The problem is that it is

often very hard to design a cost function that produces the behavior desired by the analyst. A common complaint of modelers using optimization is that it is very hard to make the model "do something" by tweaking the cost function. Sometimes, we know what we want the model to do, and we want to simply "tell it" what to do.

We have found a way to merge the strengths of both techniques. We start with an engineering cost function of the sort expected by an optimization model. This would normally be written:

$$X_t^\pi(I_t) = \arg\min_{x \in \mathcal{X}} c_t x_t$$

Assume that an element of the decision vector x_t is given by x_{adt} which is the number of times that we act on a resource with attribute a using a decision of type d. In real problems, the attribute vector a can be relatively complex (think of all the characteristics of an aircraft with a pilot moving a requirement to a destination). A *pattern* is the likelihood of a particular decision being used on a particular resource (note that we do not even consider the relationship between a decision and the state of the entire system; we restrict our patterns to relationships between decisions and the attributes of a *single* resource). However, it is very common for a pattern to be expressed at an aggregated level. Let $\hat{a}$ and $\hat{d}$ be aggregations of both the attribute vector and the decision. For example, the pattern may be that "aircraft of a particular type at a particular location should normally be sent to the following airbase." Let

$$\rho(\hat{a}, \hat{d}) = \text{The probability that decision } \hat{d} \text{ is associated with attribute } \hat{a}.$$

If $\rho(\hat{a}, \hat{d}) = 1$ then we would say that the pattern is a *rule*. In general, a pattern may simply be a probability. We incorporate patterns in the following way:

$$X_t^\pi(I_t) = \arg\min_{x \in \mathcal{X}} c_t x_t + \theta \sum_{\hat{a},\hat{d}} \left(\rho_{\hat{a}\hat{d}}(x) - \rho^h_{\hat{a}\hat{d}} \right)^2$$

where $\rho^h_{\hat{a}\hat{d}}$ represent historical priors (patterns specified to the model) and:

$$\rho_{\hat{a}\hat{d}}(x) = \frac{x_{\hat{a}\hat{d}}}{\sum_{\hat{d}} x_{\hat{a}\hat{d}}}$$

$$= \text{Fraction of time that we make decision } \hat{d}.$$

These patterns can be quite simple. Since we use a cost function, it is not necessary that they fully specify the behavior of the system. As a result, we can use patterns (defined on aggregated attribute vectors and decisions) that only have a few columns. We have found that in practice (this work has been used at a major trucking company and at two railroads) the patterns can be fairly simple. Furthermore, they may either be estimated

from historical activities (as we have done at the trucking company) or through manually specified files. We have also experimented by expressing patterns at multiple levels of aggregation.